GIT - GITHUB

O que vamos aprender?

Rodrigo: Vamos apresentar o que vamos discutir de interessante nesse curso.

Este curso é para você que trabalha com programação, ou que tem interesse em trabalhar com programação, independentemente da linguagem de programação ou da área (se vai trabalhar com back-end, front-end, mobile), pois vamos apresentar ferramentas para compartilhamento de código e para colaboração em código, que é o GitHub e também o GIT.

Nesse curso, você vai aprender o que é esse tal de GitHub, como que ele funciona, como criar uma conta nesse GitHub, criar repositórios, que são os locais onde será compartilhado o seu código, assim como trabalhar em time, compartilhando projetos com outras pessoas e essas outras pessoas também fazendo alterações no código e compartilhando com você.

Gabrielle: Você também vai aprender qual é o fluxo de trabalho dentro do GIT e GitHub.

Rodrigo: E, além disso, vamos utilizar o Prompt de Comandos e a IDE Visual Studio Code para fazer essa integração com o GIT e aprenderemos também como lidar com conflitos, já que num trabalho em equipe, eventualmente alguém pode mexer no mesmo arquivo, numa mesma linha e o GIT consegue nos auxiliar com essa parte de conflitos.

Gabrielle: Você também vai aprender sobre o histórico de versões do GIT e como alterá-lo, apagando e modificando algumas informações.

Rodrigo: Além disso, vamos apresentar algumas ferramentas adicionais do GitHub, como o Gist, para compartilhar trechos de código, como criar um README no seu repositório e o Gitignore, para ignorar determinados arquivos do seu projeto.

Há muita coisa legal que vamos discutir nesse curso, certo Gabi? Se você se interessou, venha conosco e nos encontramos na primeira aula.

02

Preparando o ambiente: projeto do curso

Próxima Atividade

Neste curso utilizaremos o projeto sorteador de número secreto, que foi desenvolvido nos cursos de lógica de programação. Você pode fazer o download do projeto inicial neste link:

Projeto inicial

Após baixar o arquivo zip, descompacte-o em algum diretório do seu computador e na sequência importe o projeto em sua IDE. No curso, utilizaremos o VSCode como IDE, sendo recomendado que você também utilize essa mesma IDE para facilitar o seu aprendizado.

Tudo pronto?

Bons estudos!

Transcrição

Gabrielle: Imagine que você finalizou o desenvolvimento de um projeto e agora deseja compartilhar o seu código com outras pessoas. Como isso pode ser feito? Será que é enviando por e-mail uma pasta compactada? Ou talvez colocando em um pendrive?

Rodrigo: Neste curso, vamos trabalhar com o projeto do número secreto. Inclusive, já abrimos o projeto na IDE do Visual Studio Code. A intenção é compartilhar o código, não o projeto.

Nos cursos de lógica de programação, vocês aprenderam a compartilhar este projeto utilizando o Vercel para fazer o que chamamos de deploy, colocando o projeto no ar. Ele gera uma URL que pode ser enviada para as pessoas acessarem o seu projeto.

Porém, não queremos compartilhar o projeto, queremos compartilhar o código. Queremos compartilhar este código HTML, CSS e JavaScript para que outras pessoas desenvolvedoras possam visualizar o código e colaborar com este projeto.

Conhecendo o GitHub

Gabrielle: Como compartilhamos esse código? Como outra pessoa pode visualizar e, às vezes, até contribuir para o código do nosso projeto?

Rodrigo: Assim como você mencionou, poderíamos enviá-lo por e-mail, compactá-lo, colocá-lo em um pendrive, entre outros.

Mas, da mesma forma que existe um site, como o Vercel, para realizar o deploy do projeto, existem também sites para compartilhar códigos de projetos de programação, onde podemos disponibilizar, deixar aberto para outras pessoas para visualizar, baixar e colaborar com o nosso código.

Há um site que é muito famoso para isso, não é mesmo, Gabi?

Gabrielle: Sim, é o GitHub, que é o site que vamos utilizar neste curso.

Rodrigo: Perfeito. Então, vamos abrir o navegador, abrir uma nova aba e entrar no site: github.com. Esse é um dos sites que oferece esse serviço de hospedagem de projetos e de código-fonte de aplicações.

Ao entrar no site, temos uma explicação de como funciona. Você consegue pesquisar alguns projetos, tem nomes de algumas empresas que o utilizam. Ele oferece esse serviço para hospedar o código do nosso projeto.

Então, fica a questão: será que é necessário pagar? Isso é de graça, Gabi?

Gabrielle: Nós podemos usar este site gratuitamente, só precisamos criar uma conta.

Criando uma conta no GitHub

Rodrigo: Para criar uma conta, caso você ainda não tenha, no canto superior direito no site do GitHub, existe um botão chamado "Sign up" (registrar-se).

Clicando nele, será exibido um formulário para você realizar um cadastro, que é como qualquer outro, pedindo seu nome, e-mail, entre outras informações.

Criaremos uma conta do zero, apenas para mostrar o passo-a-passo. Primeiro, você vai digitar o seu e-mail. Por exemplo, usaremos um e-mail fictício apenas para testar: rodrigo.alura87@gmail.com, pois os instrutores já tem uma conta no GitHub com o seus e-mails pessoais.

Ao clicar no botão de "continue" (continuar), será pedido para gerar uma senha. Nesse caso, incluiremos uma senha qualquer. Ele tem algumas validações, pedindo para você incluir uma senha forte com pelo menos 15 caracteres ou 8 caracteres incluindo um número e uma letra minúscula. Então, não pode digitar 1, 2, 3, 4, 5, 6.

Após clicar em continuar, será pedido para você digitar um username (nome de usuário).

Gabrielle: Esse nome deve ser único para te identificar dentro da plataforma.

Rodrigo: Não podemos colocar, por exemplo, rodrigo, porque provavelmente outra pessoa já criou uma conta com esse nome de usuário. Quando o nome não está disponível, aparece uma mensagem abaixo: "Username rodrigo is not available".

Vamos usar outro nome de usuário, por exemplo, colocaremos rodrigoalura87. Está disponível. Então, vamos continuar o cadastro com esse nome de usuário.

O site pergunta se você deseja receber anúncios e atualizações via e-mail. Vamos digitar "n" para indicar que não queremos. Se você desejar receber anúncios de promoções e afins, deve digitar "y".

Após clicar para continuar, a plataforma precisa verificar se não somos um robô, ou seja, algum software que está criando contas de forma aleatória.

Vamos clicar nesse botão para "Verificar", e tem um CAPTCHA um tanto peculiar. É um pouco diferente dos tradicionais. Ele exibe duas imagens, uma imagem à esquerda com um dedo apontando para alguma direção e à direita tem outro desenho de um animal de pelagem curta.

Há algumas setas que você precisa clicar para fazer esse animal apontar para a mesma direção que os dedos indicam.

Existe a opção de escutar em áudio ou reiniciar o CAPTCHA. Neste caso, vamos apenas virar o animal para a direita e clicar em "Enviar". Ele vai verificar se a resposta está correta e, então, habilitar o botão "Create account" (criar conta).

Pronto. Após criar, o Google Chrome pergunta se queremos salvar a senha no navegador. Podemos salvar.

Agora, o GitHub pede para confirmar a conta. O GitHub te envia um e-mail, fornecendo um código que você precisa colar em "Enter code".

Já temos o e-mail cadastrado aberto, você precisa aguardar um pouco e em breve chegará o e-mail do GitHub. Se for por Gmail, confira as caixas de promoções, social ou spam, pois às vezes acaba indo para lá.

Clicamos no e-mail do GitHub, vamos copiar o código para continuar o cadastro. É um código para confirmar que somos proprietários do e-mail.

Voltamos a aba do GitHub e o colamos. Após a verificação, aparecem algumas perguntas: se trabalhamos sozinhos ou se fazemos parte de uma equipe ou empresa. Vamos selecionar a opção "Just me" (apenas eu), pois trabalharemos majoritariamente sozinhos.

Em seguida, pergunta se você é um estudante ou professor. Lembra-se que, como mencionamos, este site tem um plano gratuito, mas também oferece um plano pago, com algumas opções adicionais. Então, vamos indicar que não sou nem estudante, nem professor, ou seja, marcamos "N/A".

Após clicar para continuar, ele pergunta para que você vai usar o GitHub e quais funcionalidades você tem interesse em utilizar. Tem algumas opções, mas vamos marcar apenas a primeira opção de "Collaborative coding" (programação colaborativa).

Logo abaixo, temos o botão "Continuar" e aí o GitHub apresenta o plano gratuito e o plano de times, que é pago.

E aí, vamos pagar ou vamos usar a opção gratuita?

Gabrielle: Acredito que para o nosso propósito, o plano gratuito é suficiente.

Rodrigo: Você também não precisa pagar, pode usar o plano gratuito. O plano de times, que é privado, tem um custo e, em geral, é mais voltado para grandes empresas, que têm muitas pessoas e, portanto, consegue-se usar mais recursos do GitHub. Não será o nosso caso, então vamos clicar em "Continue for free" (continue gratuitamente).

Feito isso, surge uma animação que parece que estamos navegando no espaço, não é interessante?

Agora, abre-se a página inicial do GitHub. Após você criar a conta, ou se fechar o navegador e fizer login novamente, ele te direciona para essa tela.

Agora que criamos a nossa conta no GitHub, já podemos enviar código para cá, certo?

Gabrielle: Sim, é só importante lembrar que no momento em que for criar sua conta, pode ser que esse formulário seja um pouco diferente, tenha um layout diferente, mas não se preocupe, porque esses passos serão relativamente os mesmos sempre.

Com a nossa conta criada, agora podemos prosseguir para compartilhar o nosso projeto na sequência.

Transcrição

Gabrielle: Criamos uma conta no GitHub que já está ativa e agora podemos subir o nosso projeto. Entretanto, como podemos fazer o upload desse projeto? Como organizar vários projetos no GitHub?

Rodrigo: Esta é uma questão interessante. Criamos a conta, mas e se tivermos dois, três, cinco, dez ou mesmo cem projetos? É bastante comum desenvolver vários projetos quando estamos aprendendo a programar. Queremos separá-los, porque cada projeto tem seus próprios arquivos específicos, um não tem relação com o outro.

O GitHub tem um conceito conhecido como repositório:

O repositório de um projeto é o local onde vamos guardar os códigos de um projeto específico.

Então, a ideia agora é criar um novo repositório e depois fazer o upload do nosso projeto.

Mas antes de fazer essa mudança, vamos fazer uma pequena configuração no GitHub. Vamos alterar o tema do claro para o tema escuro, para melhorar a visualização. Podemos fazer isso, clicando no canto superior direito da tela na foto do seu avatar para expandir o menu.

Em seguida, você vai até a opção de "Settings" (configurações). Nessa opção, no menu lateral esquerdo, tem a opção "Appearance" (aparência). Ele sincronizou com o sistema operacional, mas vamos mudar essa opção na combobox para "single theme" (tema único). E depois selecionar a opção "dark default" (escuro padrão).

Dica: Você pode escolher opções de alto contraste, clicando em "Dark high contrast" ou para daltonismo, clicando em "Dark Protanopia & Deuteranopia" ou "Dark Tritanopia".

Pronto, alteramos o tema para o escuro. Vamos voltar para a tela principal do GitHub, clicando no ícone do GitHub no canto superior esquerdo da tela.

Criando um repositório

Rodrigo: Para criar o repositório, podemos clicar no botão verde chamado "Create Repository" (criar repositório) na lateral esquerda da página inicial do GitHub. Ou, no ícone de "+" no menu superior, podemos ir em "Create new > New Repository".

Existem várias opções para criar um novo repositório.

Com isso, se abre um novo formulário com a conta logada já identificada. O GitHub pede para digitar o nome desse repositório. É como se fosse o nome do projeto e deve ser único.

Gabrielle: Da mesma forma que temos o nome de usuário único, também precisamos de um nome único para o nosso repositório.

Esse nome único se refere a um repositório de uma pessoa. Então, poderíamos criar um repositório na sua conta com o mesmo nome que outro repositório em minha conta.

O nome do repositório tem que ser único na sua conta.

Rodrigo: Por exemplo, poderíamos colocar projeto. Ele não vai verificar em todos os usuários do GitHub se alguém já escolheu esse nome. Ele vai verificar apenas no nosso usuário se já temos um repositório com esse nome. Se já tiver, ele não permite, caso contrário geraria um conflito de nomes.

Nesse caso, vamos criar com o nome numero-secreto, que é o nome do nosso projeto.

No próximo campo, podemos inserir uma descrição. É opcional, caso queira descrever o que é esse projeto. Vamos deixar em branco.

O próximo campo é importante. Devemos colocar se esse repositório será público (public) ou privado (private).

Gabrielle: A visibilidade do nosso repositório é definida nesse passo. Se deixarmos marcado como público, qualquer pessoa que tiver a URL do repositório conseguirá acessar o nosso projeto e visualizar os códigos.

Se deixarmos como privado, será um repositório fechado com informações mais confidenciais.

Rodrigo: Essa escolha vai depender de como você deseja compartilhar esse código.

Se for um projeto para compartilhar, montar um portfólio e você quer mostrar o código para outras pessoas, deixe-o como público. Agora, se o projeto é pessoal ou da empresa, ou seja, é restrito e outras pessoas não devem acessar o código, deixe-o como privado.

Deixaremos o repositório como público no nosso caso.

Existem algumas outras opções, como README, .gitignore, license (licença), mas não modificaremos essas opções agora. Explicaremos o que cada uma delas significa posteriormente no curso.

Para iniciar, inserimos apenas o nome e marcamos como público. Em seguida, clicamos no botão verde "Create Repository" (criar repositório) na parte inferior.

O GitHub criará o repositório e nos redirecionará para a página do repositório. Note como a URL do navegador é:

github.com/rodrigoalura87/numero-secreto

Copiar código

Onde rodrigoaluro87 é o nome do usuário que criamos na conta GitHub e numero-secreto é o nome do repositório.

Agora que já criamos o repositório, agora vamos ter que obter o código do projeto que está no nosso computador e fazer o upload para esse repositório.

Gabrielle: Correto. O GitHub até mostra um tipo de tutorial, um passo a passo, sobre como usar esses comandos e fazer o upload do nosso projeto. Veremos isso na sequência.

06

Para saber mais: repositórios privados

Próxima Atividade

A visibilidade dos repositórios no GitHub serve para controlar quem tem acesso a eles.

Repositórios públicos permitem que qualquer pessoa na internet os acesse, enquanto repositórios privados só podem ser acessados pelo dono do repositório e as pessoas que são adicionadas como colaboradoras do projeto.

Caso você não saiba o que é um colaborador em um projeto, não se preocupe. Mostraremos como isso funciona no GitHub mais adiante.

Nas contas gratuitas do GitHub existe uma condição para o uso de repositórios privados: a quantidade de pessoas colaboradoras em cada projeto é limitada a três. Para a inclusão de mais pessoas em repositórios privados, é preciso utilizar um plano pago ou manter a visibilidade como pública.

Caso queira se aprofundar mais no assunto, deixamos como recomendação a leitura das documentações a seguir:

Sobre repositórios - GitHub Docs

Definir a visibilidade do repositório - GitHub Docs

07

Para saber mais: GitHub Student Developer Pack

Próxima Atividade

Caso você seja estudante em uma instituição de ensino, você pode aproveitar os benefícios de uma conta Pro do GitHub (conta paga), além de acesso gratuito a diversas ferramentas pagas de desenvolvimento de software, através da licença para estudantes do GitHub (a GitHub Student Developer Pack).

Para obter essa licença, você precisará do seu e-mail institucional e do comprovante de matrícula. Depois, basta preencher o formulário disponível no site GitHub Education e aguardar a aprovação.

Para mais informações, você pode acessar o site do GitHub Student Developer Pack.

Transcrição

Gabrielle: Já criamos nossa conta e o repositório. Agora, como faremos essa conexão do projeto com o repositório do GitHub que criamos? Existe um tutorial, mas onde e como executamos esses comandos?

Rodrigo: Nós já fizemos a parte do GitHub, criamos a conta e o repositório. Entretanto, o projeto está no VS Code. Baixamos o projeto, importamos no VS Code e, portanto, está local no nosso computador.

O que precisamos fazer é criar ligação entre o projeto que está no computador e o repositório do GitHub.

Diferença entre Git e GitHub

Esse tutorial que apareceu após criar o repositório, é uma série de comandos que precisamos executar tanto para criar um novo repositório pela linha de comando ou fazer upload de repositório existente pela linha de comando.

São comandos, logo vamos precisar abrir o prompt de comandos do Windows.

Porém, já que estamos utilizando o Visual Studio Code, ele tem um terminal integrado. No menu superior do VS Code, clicamos na opção "Terminal > Novo terminal" (ou "Ctrl + Shift + '"). No canto inferior, é aberto o prompt de comandos.

Queremos rodar esses comandos por esse terminal integrado ao projeto. Quando abrimos o terminal pelo VS Code, ele já está na pasta do projeto aberto, que no nosso computador se chama C:\Users\Rodrigo\Desktop\numeros-secretos.

Precisamos rodar aqueles comandos na pasta do projeto. Vamos ao GitHub para copiá-los.

Se você ainda não tem um repositório local no seu computador, o GitHub fornece os seguintes comandos:

echo "# numero-secreto" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/rodrigoalura87/numero-secreto.git

git push -u origin main

Copiar código

Se você já tem um repositório existente local, outros comandos são sugeridos:

git remote add origin https://github.com/rodrigoalura87/numero-secreto.git

git branch -M main

git push -u origin main

Copiar código

O comando que começa com echo seria apenas se você quisesse criar um arquivo de exemplo do zero. Não é o nosso caso, já temos um projeto com arquivos. Então, vamos começar a partir do segundo passo.

Precisamos rodar esse comando git init. Vamos copiar esse comando com "Ctrl + C" e colar com "Ctrl + V" no terminal integrado. Vamos pressionar "Enter" e verificar o que acontece.

git init

Copiar código

git: O termo 'git' não é reconhecido como nome de cmdlet, função, arquivo de script ou programa operável. Verifique a grafia do nome ou, se um caminho tiver sido incluído, veja se o caminho está correto e tente novamente.

Gabi, não deu certo, deu um erro. Mas, o que aconteceu? Por que não funcionou?

Gabrielle: Esse erro está acontecendo porque precisamos instalar o Git no nosso computador.

Rodrigo: Exatamente, o Git é uma ferramenta que nos permite trabalhar com a parte de repositório e histórico. Mas, qual a diferença entre Git e GitHub?

Gabrielle: São ferramentas diferentes.

O GitHub é a plataforma onde vamos hospedar nosso código, enquanto o Git é a ferramenta onde conseguimos fazer o controle de versões do nosso código, registrar quais as mudanças foram feitas ao longo do tempo.

Rodrigo: São duas ferramentas separadas. O GitHub já conhecemos, e a partir de agora, precisaremos ter o Git instalado no computador. Ainda não temos essa ferramenta instalada, por isso que apareceu essa mensagem de erro.

Instalando o Git

Rodrigo: Portanto, vamos baixar e instalar o Git no computador. No navegador, vamos abrir uma nova aba e pesquisar "git download" no mecanismo de busca. O primeiro link é a página de download do site do Git.

Na página de downloads, você deve escolher o seu sistema operacional. No nosso caso, vamos clicar na opção do Windows. Depois, devemos escolher o instalador para 32 ou 64 bits - no geral é 64 bits.

São oferecidas duas opções: o instalador standalone (sozinho) e portable (portátil). O portátil não instala no computador, e, sim, cria um executável que você executa no computador.

Vamos baixar a primeira opção standalone para Windows 64 bits, pois queremos instalar de fato no computador.

Em seguida, abre-se a tela de downloads. Dependendo da sua conexão, pode demorar um pouco. Ele vai baixar um arquivo, provavelmente .exe ou .msi, para fazer a instalação no Windows. No nosso caso, foi um .exe.

Após terminar, você vai até a pasta de downloads e clica duas vezes para abrir o arquivo .exe. Não se preocupe, não haverá vírus desde que você baixe do site oficial do Git.

Vamos clicar em "sim" para permitir que o aplicativo faça alterações no dispositivo. E então vamos para a instalação.

A instalação é bem direta. Basta clicar para avançar em "Next" até concluir a instalação com "Finish". São várias opções, mas não precisa se preocupar, manteremos todas as opções padrão.

Pode surgir um alerta avisando que já existe essa pasta e perguntado se desejamos sobrescrevê-la. No nosso caso, já tínhamos uma pasta chamada Git nos arquivos de programa, por isso, vamos permitir que ele a sobrescreva.

Agora o Git será instalado no sistema operacional, e, quando ele terminar, voltaremos ao Visual Studio Code, e aquele comando deveria funcionar, certo?

Assim que o Git finaliza a instalação, ele pergunta se queremos abrir um arquivo de notas de lançamento ou abrir o Git Bash, vamos desmarcar ambas opções. Apenas clicaremos em "Finish" para finalizar.

Configurando variáveis de ambiente

Rodrigo: Vamos voltar para o terminal integrado do VS Code. Vamos rodar novamente o comando git init?

git init

Copiar código

git: O termo 'git' não é reconhecido como nome de cmdlet, função, arquivo de script ou programa operável. Verifique a grafia do nome ou, se um caminho tiver sido incluído, veja se o caminho está correto e tente novamente.

Mas, ele ainda não reconheceu. O que aconteceu? Como já tínhamos aberto esse terminal, ele não detectou que o Git foi instalado no computador. É preciso fechar essa janela do terminal, clicando no botão "Close panel" (fechar painel) no canto direito,

Novamente, vamos lá no menu superior em "Terminal > Novo terminal". O VS Code abre um novo prompt e agora deve pegar as novas instalações. Então, vamos rodar o git init.

E aí não funcionou novamente, Gabi. Isso acontece por causa do Windows. O Windows tem variáveis de ambiente que devemos configurar.

No Windows, vamos clicar no botão iniciar e buscar por "variáveis". Vamos escolher a opção "editar as variáveis de ambiente do sistema".

Na janela que se abre, vamos clicar no botão "Variáveis de Ambiente" e outra janela se abrirá com as variáveis do sistema operacional. Na parte de baixo, temos que procurar uma variável chamada Path.

Após selecioná-la, vamos clicar no botão "Editar" para abrir uma nova janela de edição. Essa variável Path tem o caminho dos programas que estão instalados no computador. Até existe um caminho para o Git definido como C:\Programs Files\Git\cmd.

Contudo, será que esse diretório está correto? Precisamos verificar.

Por isso, vamos abrir o explorador de arquivos e fazer o caminho C:\Programs Files\Git. Existe uma pasta cmd, que contém o aplicativo do Git. Na verdade, ele já adicionou a variável de ambiente, só que ainda não a reconheceu por algum motivo.

Talvez seja preciso fechar e abrir novamente o Visual Studio Code para ele detectar essa mudança.

Agora que verificamos que as variáveis estão configuradas, vamos fechar todas as janelas e fechar o Visual Studio Code.

Gabrielle: Caso no seu computador essa variável de ambiente não esteja configurada, você vai editar a variável de ambiente Path e adicionar o caminho para onde o seu Git está instalado.

Rodrigo: Perfeito. Vamos abrir o Visual Studio Code e também o terminal integrado. Vamos novamente rodar git init e dar "Enter".

Initialized empty Git repository in C:/Users/Rodrigo/Desktop/numero-secreto/.git/

Agora deu certo, pois ele mostrou a mensagem: "Repositório vazio do Git foi inicializado".

Dica: Quando for instalar o Git, verifique nas variáveis de ambiente se o caminho foi adicionado corretamente. Às vezes, não é adicionado. Nesse caso, você terá de colocar o caminho daquela pasta cmd do Git no seu computador.

E não basta apenas fechar o terminal integrado no Visual Studio Code. É necessário reiniciar o Visual Studio Code para detectar o Git.

Agora, Gabi, rodamos o primeiro comando. Na sequência, entenderemos quais são aqueles outros comandos, o que eles fazem.

09

Para saber mais: git init

Próxima Atividade

O comando git init deve ser utilizado para converter um diretório existente no computador, que geralmente é o diretório de algum projeto, em um repositório Git. No entanto, seu uso requer atenção e cuidado para evitar problemas indesejados. Vamos entender melhor o que o comando git init faz.

O que é o comando git init?

Para que possamos realizar o controle de versão de um projeto, registrando as mudanças realizadas nele ao longo do tempo, devemos, primeiramente, transformar o diretório do projeto em um repositório Git. O comando git init é utilizado para esse objetivo, devendo ser executado apenas uma vez. Quando executado, ele configura o diretório atual para ser rastreado pelo Git, inicializando um repositório vazio.

Executando o comando git init

Suponha que no seu computador exista um diretório chamado meu-projeto, que represente um projeto pessoal seu e que você deseja transformar em um repositório do Git. Para isso, você pode abrir esse diretório no VSCode, abrir uma janela do terminal e executar o comando git init. A saída será algo como:

Terminal do VSCode com o comando git init sendo executado no diretório "~/Projetos/alura/meu-projeto" e recebendo como saída a mensagem "Initialized empty Git repository in /home/rodrigo/Projetos/alura/meu-projeto/.git/"

Repare na imagem anterior que o comando git init foi executado no terminal do VSCode. Observe também que no terminal é indicado qual o diretório no qual o comando foi executado, que no exemplo foi em: ~/Projetos/alura/meu-projeto. É importante se atentar a isso, pois o comando git init transforma o diretório atual em um repositório do Git, logo ele deve ser executado dentro do diretório do projeto e não em outros diretórios do computador.

Ao executar o comando, note que a saída no terminal foi a mensagem Initialized empty Git repository in /home/rodrigo/Projetos/alura/meu-projeto/.git/. Essa mensagem indica que o comando foi executado corretamente e um repositório local do Git foi criado com sucesso nesse diretório. A partir desse ponto, já podemos trabalhar no projeto, adicionando arquivos, realizando modificações e registrando as mudanças no Git.

Cuidados com o comando git init

Aprendemos que o comando git init serve para criar um novo repositório Git e por isso deve ser executado apenas uma única vez. Ou seja, se um diretório já for um repositório Git, não faz sentido rodar novamente o comando git init. Esse é um erro bastante comum de ser cometido.

Se você executar o comando git init em um diretório que já foi inicializado como um repositório Git, a seguinte mensagem será exibida:

Reinitialized existing Git repository in /home/rodrigo/Projetos/alura/meu-projeto/.git/

Copiar código

Isso indica que o Git reinicializou um repositório já existente, ou seja, o comando git init foi executado em um diretório que já era um repositório Git.

Caso você tenha cometido esse equívoco, não precisa se preocupar, pois todo o histórico de mudanças e commits no projeto não será apagado. O Git detecta que o diretório já era um repositório Git e com isso o comando não tem efeito nenhum.

Na dúvida, antes de executar o comando git init, execute primeiramente o comando git status. Se aparecer a mensagem fatal: not a git repository (or any of the parent directories): .git, isso significa que o diretório atual não é um repositório Git e você pode então executar o comando git init.

Configuração rápida - se você já fez esse tipo de coisa antes

ou

https://github.com/Agapho2/numero-secreto.git

Comece criando um novo arquivo ou carregando um arquivo existente. Recomendamos que todos os repositórios incluam um README, LICENSE, e .gitignore.

… ou criar um novo repositório na linha de comando

echo "# numero-secreto" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/Agapho2/numero-secreto.git

git push -u origin main

… ou envie um repositório existente a partir da linha de comando

git remote add origin https://github.com/Agapho2/numero-secreto.git

git branch -M main

git push -u origin main

09

Para saber mais: git init

Próxima Atividade

O comando git init deve ser utilizado para converter um diretório existente no computador, que geralmente é o diretório de algum projeto, em um repositório Git. No entanto, seu uso requer atenção e cuidado para evitar problemas indesejados. Vamos entender melhor o que o comando git init faz.

O que é o comando git init?

Para que possamos realizar o controle de versão de um projeto, registrando as mudanças realizadas nele ao longo do tempo, devemos, primeiramente, transformar o diretório do projeto em um repositório Git. O comando git init é utilizado para esse objetivo, devendo ser executado apenas uma vez. Quando executado, ele configura o diretório atual para ser rastreado pelo Git, inicializando um repositório vazio.

Executando o comando git init

Suponha que no seu computador exista um diretório chamado meu-projeto, que represente um projeto pessoal seu e que você deseja transformar em um repositório do Git. Para isso, você pode abrir esse diretório no VSCode, abrir uma janela do terminal e executar o comando git init. A saída será algo como:

Terminal do VSCode com o comando git init sendo executado no diretório "~/Projetos/alura/meu-projeto" e recebendo como saída a mensagem "Initialized empty Git repository in /home/rodrigo/Projetos/alura/meu-projeto/.git/"

Repare na imagem anterior que o comando git init foi executado no terminal do VSCode. Observe também que no terminal é indicado qual o diretório no qual o comando foi executado, que no exemplo foi em: ~/Projetos/alura/meu-projeto. É importante se atentar a isso, pois o comando git init transforma o diretório atual em um repositório do Git, logo ele deve ser executado dentro do diretório do projeto e não em outros diretórios do computador.

Ao executar o comando, note que a saída no terminal foi a mensagem Initialized empty Git repository in /home/rodrigo/Projetos/alura/meu-projeto/.git/. Essa mensagem indica que o comando foi executado corretamente e um repositório local do Git foi criado com sucesso nesse diretório. A partir desse ponto, já podemos trabalhar no projeto, adicionando arquivos, realizando modificações e registrando as mudanças no Git.

Cuidados com o comando git init

Aprendemos que o comando git init serve para criar um novo repositório Git e por isso deve ser executado apenas uma única vez. Ou seja, se um diretório já for um repositório Git, não faz sentido rodar novamente o comando git init. Esse é um erro bastante comum de ser cometido.

Se você executar o comando git init em um diretório que já foi inicializado como um repositório Git, a seguinte mensagem será exibida:

Reinitialized existing Git repository in /home/rodrigo/Projetos/alura/meu-projeto/.git/

Copiar código

Isso indica que o Git reinicializou um repositório já existente, ou seja, o comando git init foi executado em um diretório que já era um repositório Git.

Caso você tenha cometido esse equívoco, não precisa se preocupar, pois todo o histórico de mudanças e commits no projeto não será apagado. O Git detecta que o diretório já era um repositório Git e com isso o comando não tem efeito nenhum.

Na dúvida, antes de executar o comando git init, execute primeiramente o comando git status. Se aparecer a mensagem fatal: not a git repository (or any of the parent directories): .git, isso significa que o diretório atual não é um repositório Git e você pode então executar o comando git init.

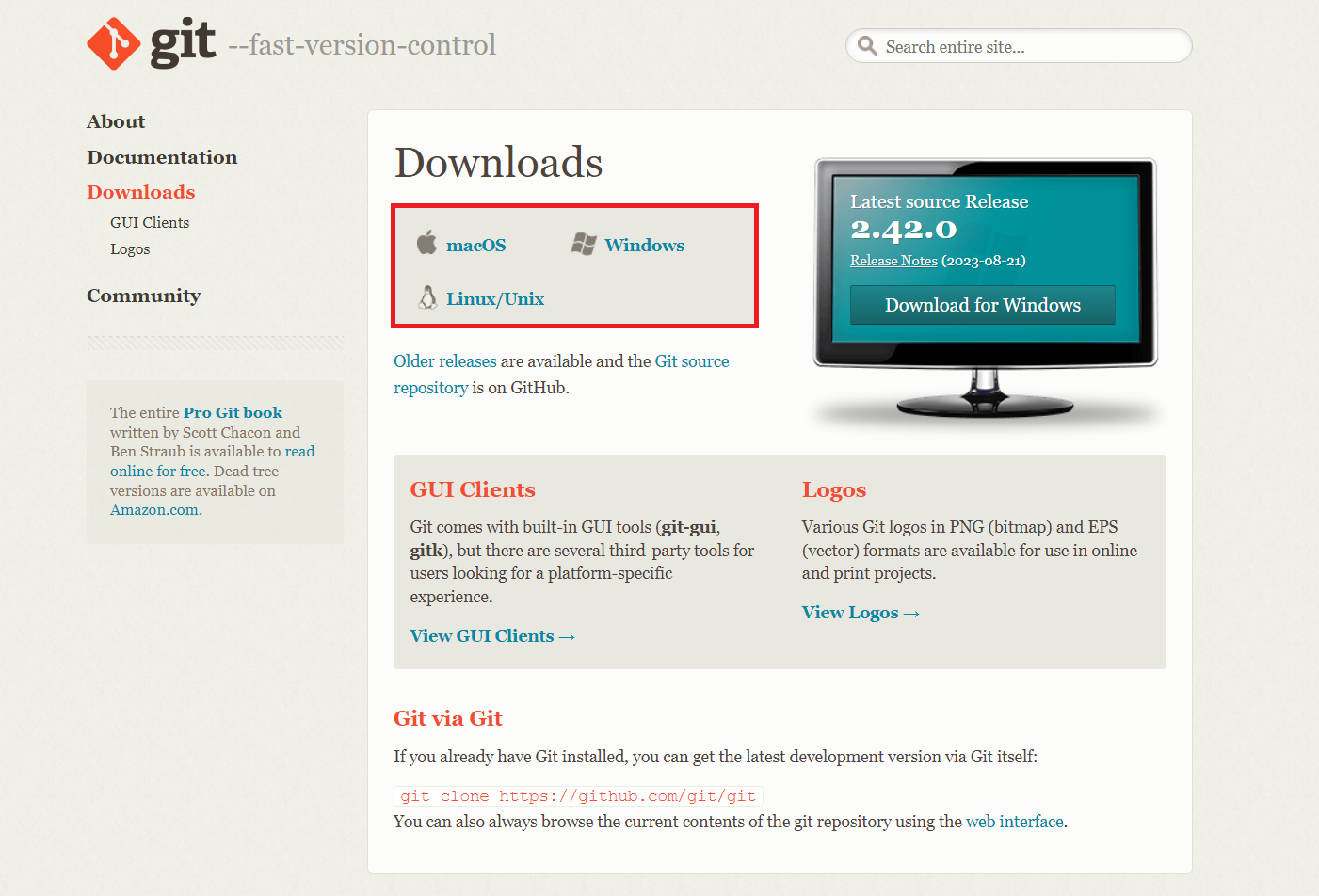
**10Faça como eu fiz: instalação do Git**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/144999/next)

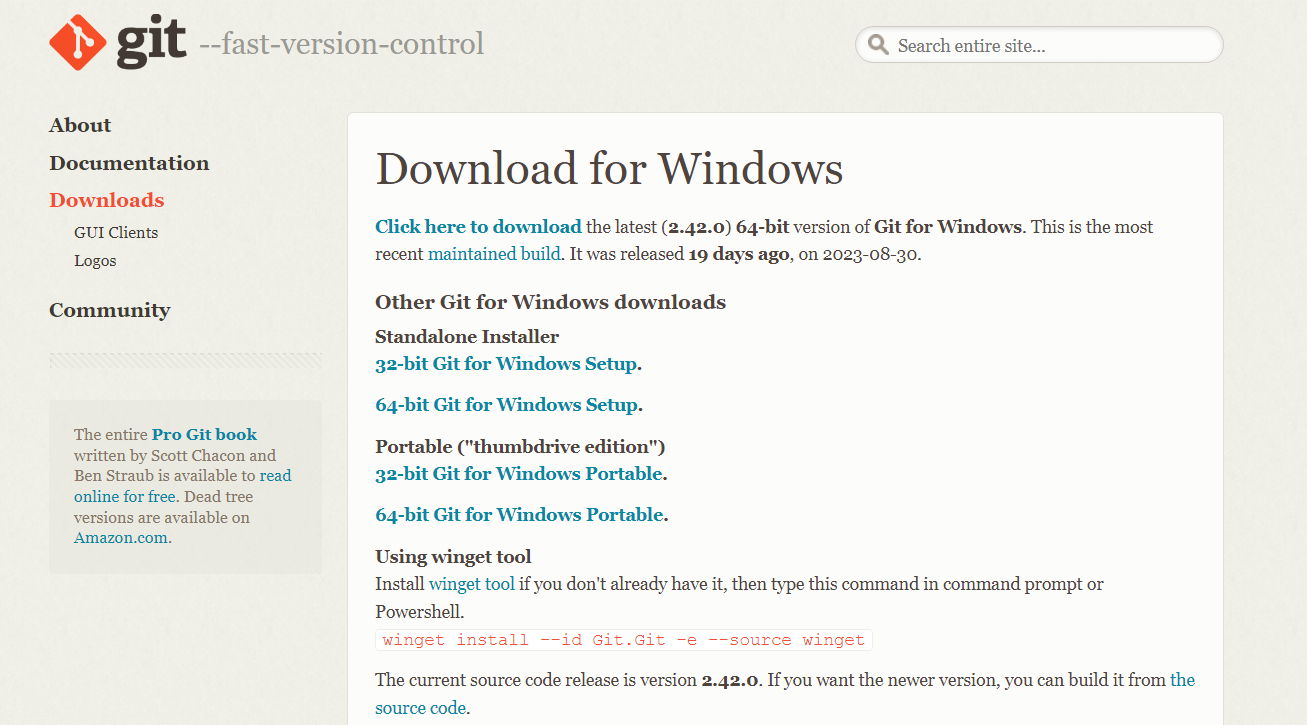
Agora é com você! Caso ainda não tenha o Git instalado em seu computador, faça o download e instalação conforme demonstrado no vídeo anterior.

**Windows**

Para instalar o Git em seu computador, você deve acessar o [site de downloads do Git](https://git-scm.com/downloads). Em seguida, você deve clicar no botão referente ao seu sistema operacional. No nosso caso, clicamos em **Windows**.

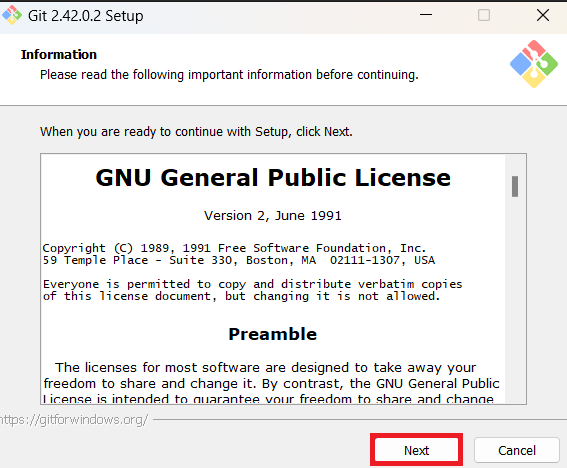


No site do Git será aberta uma lista com diversas opções de download. Vamos clicar na opção *64-bit Git for Windows Setup*. Assim, baixamos um arquivo com a extensão *.exe*.

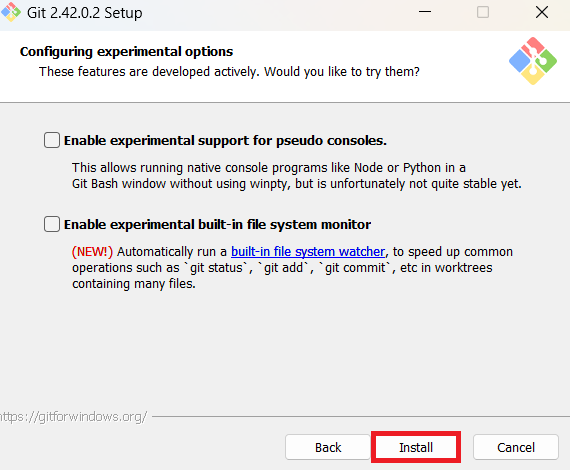


Você deve clicar no arquivo baixado para iniciar a instalação. Provavelmente uma janela será aberta perguntando se você deseja permitir que esse programa faça alterações em seu computador; caso esse cenário ocorra, você deve clicar em *Sim* para continuar.

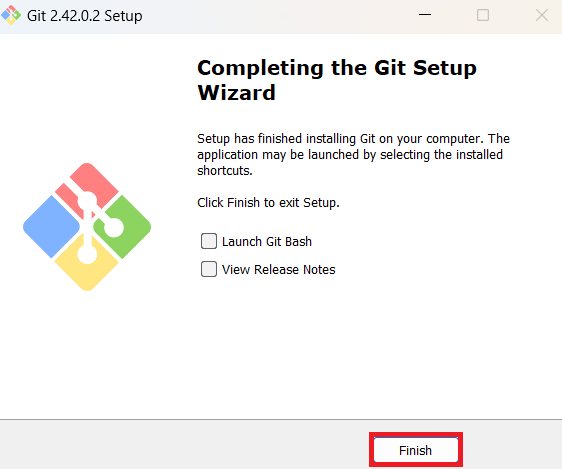
Na janela de instalação, você deve prosseguir clicando nos botões de *Next*. Neste momento, não é necessário realizar nenhuma alteração nas configurações padrão da instalação do Git.



Após uma sequência de cliques no botão *Next*, você deve clicar no botão *Install*, para instalar o Git.



Por fim, clique no botão *Finish*.



Assim, o Git estará instalado em seu computador.

**MacOS**

Existem várias opções para instalar o Git no macOS, mas o próprio git indica algumas soluções.

Ao clicar em macOS na página de [download](https://git-scm.com/downloads), você pode conferir a instalação.

O **[Homebrew](https://brew.sh/" \t "_blank)** é um gerenciador de pacotes para o macOS que simplifica a instalação de software.

Abra o Terminal e cole o comando que o site oferece para instalar o Homebrew.

**Transcrição**

**Gabrielle**: Já instalamos o Git e inicializamos o repositório local com o comando git init. Mas agora, como podemos prosseguir para **conectar esse repositório local ao repositório remoto** no GitHub.

**Rodrigo**: Isso é muito importante. O Git possui esse conceito de **repositório local e repositório remoto**. Quando criamos o repositório no **GitHub**, o repositório que se encontra lá é denominado repositório remoto. No entanto, em nosso **computador**, também precisamos criar um repositório, que será o local.

Esses dois repositórios não estão conectados. Agora que instalamos o Git e criamos o repositório local, podemos seguir o tutorial que foi apresentado para estabelecer essa conexão entre o meu repositório local e o repositório remoto no GitHub.

**Adicionar arquivos e vincular repositório**

**Rodrigo**: Vamos retornar ao site do GitHub. Já executamos o comando git init. O próximo comando que aparece no tutorial é git add, que mostra como exemplo um arquivo README.md que não possui existe no nosso projeto. Ao invés disso, precisamos adicionar os arquivos do nosso projeto.

Vamos copiar o comando git add e o colar no terminal integrado do VS Code. No entanto, precisamos adicionar cada um dos arquivos do projeto. Mas, temos vários arquivos neste projeto, como o app.js, index.html e outros. Precisaremos adicionar cada um manualmente?

**Gabrielle**: Temos a opção de adicionar manualmente, informando o nome de cada um dos arquivos, mas às vezes pode ser um pouco trabalhoso, especialmente quando temos um projeto grande com muitos arquivos.

Para facilitar esse processo, temos um atalho no comando do Git, onde podemos **adicionar todos os arquivos de uma só vez**. É o comando git add .. Esse ponto vai adicionar todos os arquivos do repositório.

git **add** .

Copiar código

**Rodrigo**: Após pressionar "Enter", apareceram alguns alertas, porque existem algumas diferenças entre Windows e Linux, dependendo do sistema operacional que você estiver utilizando. No entanto, são só alertas e não vão afetar nada.

Retornando ao GitHub, o próximo comando que encontramos é o git commit -m e temos que inserir uma mensagem entre aspas. Copiamos até o -m, porque vamos alterar a mensagem. Colocaremos a mensagem projeto inicial entre aspas duplas.

git **commit** -m "projeto inicial"

Copiar código

Estes comandos, add e commit, foram somente copiados e executados, mas nas próximas aulas, vamos explicar detalhadamente o que cada um faz. No momento, o objetivo é apenas conectar os dois repositórios: o local no computador com o remoto no GitHub.

*Author identity unkown*

Após pressionar "Enter", houve um erro ao tentar fazer o *commit*.

**Gabrielle**: Apareceu uma mensagem indicando que a identidade do autor desse *commit* é desconhecida. Precisamos fazer uma configuração para informar quem é a pessoa responsável por esse *commit*.

No terminal, é exibida uma dica indicando quais comandos devemos usar para fazer essa configuração:

git config --**global** user.email "your@example.com"

git config --**global** user.name "Your Name"

Copiar código

**Rodrigo**: No GitHub, criamos uma nova conta e fiz o login. Portanto, o GitHub sabe quem somos, porque estamos logados no navegador. No entanto, instalamos o Git no computador e o Git não sabe quem somos, pois não possui vínculo direto com o GitHub.

Esses comandos são justamente para nos identificar ao Git no computador.

Copiamos apenas comando git config --global user.email, mas só até email. Vamos colar no terminal e incluir o e-mail que configuramos no GitHub. No nosso caso, é o rodrigo.alura87@gmail.com entre aspas duplas.

git config --**global** user.email "rodrigo.alura87@gmail.com"

Copiar código

**Atenção**: Você deve substituir pelo e-mail usado para criar sua conta no GitHub!

Após pressionar "Enter", vamos utilizar a seta para cima para resgatar o último comando digitado. Alteraremos o user.email para user.name, pois além do e-mail, o Git também precisa saber o seu nome.

Entre as aspas, inserimos o Rodrigo Ferreira e apertamos "Enter". Você deverá colocar seu nome e sobrenome.

git config --**global** user.name "Rodrigo Ferreira"

Copiar código

Pronto. Realizamos a configuração de maneira global, devido a esse parâmetro --global. Agora, neste computador, qualquer repositório que criamos, o Git já nos reconhece.

**Gabrielle**: Agora, podemos prosseguir e tentar realizar esse *commit*.

**Rodrigo**: Vamos apertar novamente a seta para cima três vezes para voltar para o comando de *commit* e dar um "Enter". Agora o *commit* foi realizado.

[master (root-commit) 250c665] projeto inicial

10 files changed, 334 insertions (+)

Ele **registrou esses arquivos no nosso repositório local**. Ótimo, vamos voltar para o tutorial no GitHub.

O próximo comando, git branch -M main. Vamos copiar, colar e executar. Não deu nenhum alerta.

git branch -M **main**

Copiar código

E o próximo comando é o principal, pois é o comando que vai fazer o **vínculo do repositório do GitHub com o meu repositório local**, que é esse git remote add.

**Gabrielle**: Sim, após executar esse comando, vamos conseguir que esse repositório que temos localmente em nosso computador esteja conectado com esse repositório do GitHub.

**Rodrigo**: Vamos apenas fazer uma alteração, porque por padrão ele vem com o protocolo HTTPS. Mas acima, onde temos instruções para o *Quick Setup* (configuração rápida), tem um botão chamado "SSH". Se clicamos neste botão, o GitHub troca para o protocolo para o SSH.

Geralmente, o **protocolo SSH** é o mais recomendado para fazer esse ligação com o GitHub.

O comando git remote add origin já foi alterado, e agora a URL não está com http://. Ela está usando o formato SSH. Vamos copiar e colar no terminal integrado:

git remote add origin git**@github**.com:rodrigoalura87/numero-secreto.git

Copiar código

Lembre-se que a URL do GitHub vai variar de acordo com o nome do seu usuário e projeto!

**Gerar chave SSH**

Por fim, temos o comando para enviar esses arquivos, ou seja, esse *commit* que fizemos localmente para o GitHub com o comando git push. O remote add não envia automaticamente, ele só faz a ligação.

git push -u origin **main**

Copiar código

Vamos copiar e colar esse último comando. Com isso, ele vai tentar sincronizar, de fato, com o GitHub. Vamos conferir se vai funcionar.

The authenticity of host 'GitHub.com (20.201.28.151)' can't be established

Deu outro erro! Foi um erro de autenticação. Afinal, instalamos o Git no computador, colocamos o e-mail e nome, mas como é que o GitHub vai saber que somos nós mesmos - e não é outra pessoa que está usando nosso nome e e-mail?

De alguma forma, precisamos fazer essa configuração de segurança para ele saber que neste computador, é, de fato, o Rodrigo com e-mail específico e está vinculado com a conta do Rodrigo no GitHub.

Como copiamos o protocolo SSH, o Git já nos dá a sugestão de **gerar uma chave SSH** no computador porque ainda não temos uma. Ele faz uma pergunta que precisamos responder com "sim" ou "não".

Are you sure you want to continue connecting (yes/no/[fingerprint])?

Em inglês, ele pergunta: quer continuar a conexão? Respondemos com um yes (sim). Feito isso, ele manda uma mensagem dizendo que não reconhece esse computador, portanto, não está autorizado para se conectar no GitHub.

Warning: Permanently added 'github.com' (ED25519) to the list of known hosts

git@github.com: Permission denied (publickey)

Isso é importante para garantir a segurança. Ou seja, vamos ter que fazer essa configuração no GitHub.

**Gabrielle**: Isso mesmo. Vamos ao GitHub finalizar essa configuração.

**Rodrigo**: Essas são configurações iniciais, pois como acabamos de instalar o Git, não tem nada disso configurado. Será apenas uma vez que precisamos fazer todos esses comandos.

De volta ao GitHub, no canto superior direito da página, vamos clicar no ícone do avatar para abrir o menu lateral e selecionar a opção "*Settings*". Assim, entraremos nas configurações da nossa conta e não do repositório.

Na página de configurações da nossa conta, no menu à esquerda, existe a opção "SSH and GPG keys" onde podemos fazer a configuração do SSH.

Em seguida, vamos clicar no botão verde "*New SSH key*" (nova chave SSH). Ao clicar nesse botão, devemos colocar um título. É como se estivéssemos criando uma chave para conectar esse nosso computador com o GitHub. Vamos colocar notebook pessoal, apenas para identificar que chave é essa.

Ele também solicita um tipo, nesse caso, é uma chave de autenticação (*Authentication key*). Além disso, nos orienta a colar a chave. Mas, de onde vamos conseguir essa chave, Gabi?

**Gabrielle**: Devemos gerar essa chave em nosso computador.

**Rodrigo**: Assim, no nosso computador, teremos que acessar essa chave que foi gerada com o SSH.

No terminal integrado, quando pressionamos "yes", ele deve ter criado uma chave que fica salva dentro do computador na pasta do usuário. Podemos tanto fazer isso pelo terminal ou navegar até a pasta, o que é mais fácil.

Portanto, vamos abrir o explorador de arquivos e entrar na pasta do usuário. Nesse casso, a pasta Rodrigo. Agora, vamos procurar uma pasta oculta chamada .ssh. Entrando nela, há um arquivo chamado known\_hosts, que são os hosts conhecidos, mas a chave não foi gerada.

Para gerar a chave, existe um comando que temos que copiar. Inclusive, existe um tutorial no próprio site do GitHub, que vamos disponibilizar no curso como uma atividade. Ele tem um tutorial ensinando [como gerar uma chave SSH](https://docs.github.com/pt/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent) para cada sistema operacional. No nosso caso, é o Windows.

Nessa documentação, temos a explicação do que é e como gerar uma chave SSH. Vamos copiar o comando sugerido ssh-keygen -t ed25519 -C seguido do e-mail da conta do GitHub.

No terminal do Visual Studio Code, vamos colá-lo. Porém, temos que alterar o e-mail de exemplo para o nosso e-mail, que é rodrigo.alura87@gmail.com. Você deve substituir com seu e-mail pessoal.

ssh-keygen -t ed25519 -C "rodrigo.alura87@gmail.com"

Copiar código

Após apertar "Enter", ele informa que vai gerar um par de chaves SSH. E nos pergunta onde queremos salvar esse arquivo.

Enter file in which to save the key (C:\User\Rodrigo/.ssh/id\_ed25519):

Você pode simplesmente pressionar "Enter" para salvá-lo na própria pasta SSH, que é a pasta padrão. Ele pergunta se você quer digitar uma senha.

Enter passphrase (empty for no passphrase):

Se digitamos uma senha, toda vez que eu for sincronizar com o GitHub, ele vai nos pedir essa senha. Por isso, vamos apenas pressionar "Enter" para não ter uma senha. Depois, devemos digitar a mesma senha novamente, damos "Enter" novamente.

Pronto. Agora a chave foi gerada.

Vamos até a pasta "Rodrigo > .ssh". Agora, contém dois arquivo chamamos id\_ed25519, um com a chave privada, outro com a chave pública. Devemos abrir o da chave pública, vamos conferir.

Vamos dar dois cliques e mandar abrir com o bloco de notas. Estava escrito *private key*, era a chave privada.

Vamos abrir o outro arquivo. Podemos clicar com o botão direito e selecionar "Abrir com" e escolher o aplicativo de bloco de notas. Agora, temos o código da chave começando com ssh ed-25519 que precisamos copiar e colar no site do GitHub.

No campo *key*, vamos colá-lo e apertar o botão "*Add SSH Key*" para adicionar a chave SSH. O GitHub já adicionou e listou a chave do notebook pessoal em "SSH keys".

Agora o GitHub sabe que neste notebook tem uma chave e já nos autenticamos, pois geramos essa chave com o nosso e-mail. Assim, o GitHub confia neste computador. Desse modo, conseguiremos sincronizar com a nossa conta no GitHub.

**Sincronizar repositórios**

**Rodrigo**: Vamos voltar para o Visual Studio Code e tentar fazer novamente aquele git push:

git push -u origin **main**

Copiar código

Vamos conferir se agora vai funcionar.

Total 14 (delta 0), reused 0 (delta 0), pack-reused 0

To github.com: rodrigoalura87/numero-secreto.git

* [new branch) main -> main

branch 'main' set up to track 'origin/main'.

Finalmente, o upload dos arquivos e do *commit* para o nosso repositório no GitHub foi bem-sucedido, ao que tudo indica. Será que está sincronizado, Gabi?

**Gabrielle**: Sim, aparentemente funcionou, mas vamos abrir nosso repositório no navegador e atualizar a página para verificar se encontramos nosso projeto lá?

**Rodrigo**: No navegador, precisamos voltar para a página do repositório. Basta clicar no ícone do avatar e acessar "*Your repositories*" (seus repositórios) e escolher o numero-secreto. Em seguida, vamos atualizar a página (atalho "F5") e o projeto apareceu.

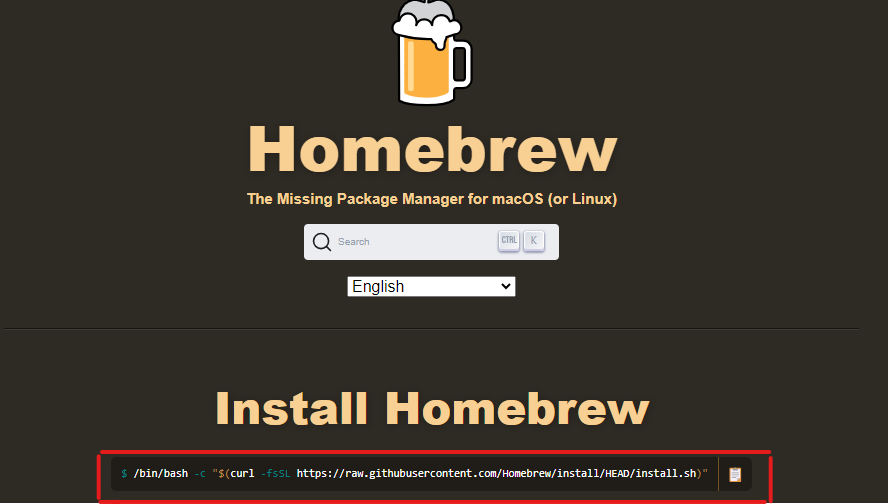
O tutorial que estava antes sumiu, porque agora tudo já está sincronizado e os arquivos estão sincronizados no repositório remoto.

Temos o index.html, o app.js e o sistema identificou o *commit* e usuário, o rodrigoalura87. Podemos navegar pelos arquivos e ter acesso ao código-fonte.

Logo, nosso repositório está sincronizado e na barra de endereços temos a URL do repositório. No nosso caso, é github.com/rodrigoalura87/numero-secreto.

Podemos compartilhar esse link com outras pessoas e elas podem acessar o repositório, inclusive você, Gabi. Eu posso te enviar este repositório e fica aí essa pergunta: será que você conseguirá baixar o projeto no seu computador?

**Gabrielle**: O GitHub nos proporciona essa possibilidade de outras pessoas não apenas acessarem nosso código, mas também contribuírem para ele. Vamos explorar isso em seguida.



/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

Copiar código

Pressione "Return" para executar o script. O Terminal pode solicitar que você insira sua senha. Digite sua senha e pressione "Return" novamente.

Após a conclusão da instalação, precisamos adicionar o brew no PATH. O terminal fornecerá instruções sobre como adicionar o diretório do Homebrew ao seu PATH. Normalmente, as instruções serão semelhantes a:

echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> /Users/seu\_usuario/.zprofile

'eval "$(/opt/homebrew/bin/brew shellenv)"'

Copiar código

Copie um de cada vez e execute no terminal.

Lembre-se de modificar o seu\_usuario pelo seu nome de usuário!

Para garantir que o Homebrew foi instalado corretamente, digite o seguinte comando:

brew --version

Copiar código

Feche o Terminal, abra-o novamente e digite o seguinte comando para instalar o git:

$ brew install git

Copiar código

Pressione "Return" para iniciar o processo de instalação.

Após a conclusão da instalação, verifique se o Git foi instalado corretamente usando o comando:

git --version

Copiar código

Pronto. Você concluiu a instalação do Git no macOS usando o Homebrew!

**Linux**

A instalação do Git no Linux pode variar um pouco dependendo da distribuição que você está utilizando. No entanto, a maioria das distribuições Linux oferece o Git nos repositórios padrão, o que torna a instalação bastante simples.

**Ubuntu e derivados do Debian**

Para instalar o Git no Ubuntu ou em distribuições baseadas em Debian, como o Debian em si, você pode usar o gerenciador de pacotes apt. Abra o terminal e execute o seguinte comando:

sudo apt install git

Copiar código

**Fedora**

Para instalar o Git no Fedora, você pode usar o gerenciador de pacotes dnf. Abra o terminal e execute o seguinte comando:

sudo dnf install git

Copiar código

**Arch Linux e derivados**

No Arch Linux e em distribuições baseadas nele, como Manjaro, você pode usar o pacman para instalar o Git. No terminal, execute o seguinte comando:

sudo pacman -S git

Copiar código

Após a conclusão da instalação, você pode verificar se o Git foi instalado corretamente digitando o seguinte comando no terminal:

git --version

Copiar código

Isso deve exibir a versão do Git que foi instalada.

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**Transcrição**

**Gabrielle**: Já instalamos o Git e inicializamos o repositório local com o comando git init. Mas agora, como podemos prosseguir para **conectar esse repositório local ao repositório remoto** no GitHub.

**Rodrigo**: Isso é muito importante. O Git possui esse conceito de **repositório local e repositório remoto**. Quando criamos o repositório no **GitHub**, o repositório que se encontra lá é denominado repositório remoto. No entanto, em nosso **computador**, também precisamos criar um repositório, que será o local.

Esses dois repositórios não estão conectados. Agora que instalamos o Git e criamos o repositório local, podemos seguir o tutorial que foi apresentado para estabelecer essa conexão entre o meu repositório local e o repositório remoto no GitHub.

**Adicionar arquivos e vincular repositório**

**Rodrigo**: Vamos retornar ao site do GitHub. Já executamos o comando git init. O próximo comando que aparece no tutorial é git add, que mostra como exemplo um arquivo README.md que não possui existe no nosso projeto. Ao invés disso, precisamos adicionar os arquivos do nosso projeto.

Vamos copiar o comando git add e o colar no terminal integrado do VS Code. No entanto, precisamos adicionar cada um dos arquivos do projeto. Mas, temos vários arquivos neste projeto, como o app.js, index.html e outros. Precisaremos adicionar cada um manualmente?

**Gabrielle**: Temos a opção de adicionar manualmente, informando o nome de cada um dos arquivos, mas às vezes pode ser um pouco trabalhoso, especialmente quando temos um projeto grande com muitos arquivos.

Para facilitar esse processo, temos um atalho no comando do Git, onde podemos **adicionar todos os arquivos de uma só vez**. É o comando git add .. Esse ponto vai adicionar todos os arquivos do repositório.

git **add** .

Copiar código

**Rodrigo**: Após pressionar "Enter", apareceram alguns alertas, porque existem algumas diferenças entre Windows e Linux, dependendo do sistema operacional que você estiver utilizando. No entanto, são só alertas e não vão afetar nada.

Retornando ao GitHub, o próximo comando que encontramos é o git commit -m e temos que inserir uma mensagem entre aspas. Copiamos até o -m, porque vamos alterar a mensagem. Colocaremos a mensagem projeto inicial entre aspas duplas.

git **commit** -m "projeto inicial"

Copiar código

Estes comandos, add e commit, foram somente copiados e executados, mas nas próximas aulas, vamos explicar detalhadamente o que cada um faz. No momento, o objetivo é apenas conectar os dois repositórios: o local no computador com o remoto no GitHub.

*Author identity unkown*

Após pressionar "Enter", houve um erro ao tentar fazer o *commit*.

**Gabrielle**: Apareceu uma mensagem indicando que a identidade do autor desse *commit* é desconhecida. Precisamos fazer uma configuração para informar quem é a pessoa responsável por esse *commit*.

No terminal, é exibida uma dica indicando quais comandos devemos usar para fazer essa configuração:

git config --**global** user.email "your@example.com"

git config --**global** user.name "Your Name"

Copiar código

**Rodrigo**: No GitHub, criamos uma nova conta e fiz o login. Portanto, o GitHub sabe quem somos, porque estamos logados no navegador. No entanto, instalamos o Git no computador e o Git não sabe quem somos, pois não possui vínculo direto com o GitHub.

Esses comandos são justamente para nos identificar ao Git no computador.

Copiamos apenas comando git config --global user.email, mas só até email. Vamos colar no terminal e incluir o e-mail que configuramos no GitHub. No nosso caso, é o rodrigo.alura87@gmail.com entre aspas duplas.

git config --**global** user.email "rodrigo.alura87@gmail.com"

Copiar código

**Atenção**: Você deve substituir pelo e-mail usado para criar sua conta no GitHub!

Após pressionar "Enter", vamos utilizar a seta para cima para resgatar o último comando digitado. Alteraremos o user.email para user.name, pois além do e-mail, o Git também precisa saber o seu nome.

Entre as aspas, inserimos o Rodrigo Ferreira e apertamos "Enter". Você deverá colocar seu nome e sobrenome.

git config --**global** user.name "Rodrigo Ferreira"

Copiar código

Pronto. Realizamos a configuração de maneira global, devido a esse parâmetro --global. Agora, neste computador, qualquer repositório que criamos, o Git já nos reconhece.

**Gabrielle**: Agora, podemos prosseguir e tentar realizar esse *commit*.

**Rodrigo**: Vamos apertar novamente a seta para cima três vezes para voltar para o comando de *commit* e dar um "Enter". Agora o *commit* foi realizado.

[master (root-commit) 250c665] projeto inicial

10 files changed, 334 insertions (+)

Ele **registrou esses arquivos no nosso repositório local**. Ótimo, vamos voltar para o tutorial no GitHub.

O próximo comando, git branch -M main. Vamos copiar, colar e executar. Não deu nenhum alerta.

git branch -M **main**

Copiar código

E o próximo comando é o principal, pois é o comando que vai fazer o **vínculo do repositório do GitHub com o meu repositório local**, que é esse git remote add.

**Gabrielle**: Sim, após executar esse comando, vamos conseguir que esse repositório que temos localmente em nosso computador esteja conectado com esse repositório do GitHub.

**Rodrigo**: Vamos apenas fazer uma alteração, porque por padrão ele vem com o protocolo HTTPS. Mas acima, onde temos instruções para o *Quick Setup* (configuração rápida), tem um botão chamado "SSH". Se clicamos neste botão, o GitHub troca para o protocolo para o SSH.

Geralmente, o **protocolo SSH** é o mais recomendado para fazer esse ligação com o GitHub.

O comando git remote add origin já foi alterado, e agora a URL não está com http://. Ela está usando o formato SSH. Vamos copiar e colar no terminal integrado:

git remote add origin git**@github**.com:rodrigoalura87/numero-secreto.git

Copiar código

Lembre-se que a URL do GitHub vai variar de acordo com o nome do seu usuário e projeto!

**Gerar chave SSH**

Por fim, temos o comando para enviar esses arquivos, ou seja, esse *commit* que fizemos localmente para o GitHub com o comando git push. O remote add não envia automaticamente, ele só faz a ligação.

git push -u origin **main**

Copiar código

Vamos copiar e colar esse último comando. Com isso, ele vai tentar sincronizar, de fato, com o GitHub. Vamos conferir se vai funcionar.

The authenticity of host 'GitHub.com (20.201.28.151)' can't be established

Deu outro erro! Foi um erro de autenticação. Afinal, instalamos o Git no computador, colocamos o e-mail e nome, mas como é que o GitHub vai saber que somos nós mesmos - e não é outra pessoa que está usando nosso nome e e-mail?

De alguma forma, precisamos fazer essa configuração de segurança para ele saber que neste computador, é, de fato, o Rodrigo com e-mail específico e está vinculado com a conta do Rodrigo no GitHub.

Como copiamos o protocolo SSH, o Git já nos dá a sugestão de **gerar uma chave SSH** no computador porque ainda não temos uma. Ele faz uma pergunta que precisamos responder com "sim" ou "não".

Are you sure you want to continue connecting (yes/no/[fingerprint])?

Em inglês, ele pergunta: quer continuar a conexão? Respondemos com um yes (sim). Feito isso, ele manda uma mensagem dizendo que não reconhece esse computador, portanto, não está autorizado para se conectar no GitHub.

Warning: Permanently added 'github.com' (ED25519) to the list of known hosts

git@github.com: Permission denied (publickey)

Isso é importante para garantir a segurança. Ou seja, vamos ter que fazer essa configuração no GitHub.

**Gabrielle**: Isso mesmo. Vamos ao GitHub finalizar essa configuração.

**Rodrigo**: Essas são configurações iniciais, pois como acabamos de instalar o Git, não tem nada disso configurado. Será apenas uma vez que precisamos fazer todos esses comandos.

De volta ao GitHub, no canto superior direito da página, vamos clicar no ícone do avatar para abrir o menu lateral e selecionar a opção "*Settings*". Assim, entraremos nas configurações da nossa conta e não do repositório.

Na página de configurações da nossa conta, no menu à esquerda, existe a opção "SSH and GPG keys" onde podemos fazer a configuração do SSH.

Em seguida, vamos clicar no botão verde "*New SSH key*" (nova chave SSH). Ao clicar nesse botão, devemos colocar um título. É como se estivéssemos criando uma chave para conectar esse nosso computador com o GitHub. Vamos colocar notebook pessoal, apenas para identificar que chave é essa.

Ele também solicita um tipo, nesse caso, é uma chave de autenticação (*Authentication key*). Além disso, nos orienta a colar a chave. Mas, de onde vamos conseguir essa chave, Gabi?

**Gabrielle**: Devemos gerar essa chave em nosso computador.

**Rodrigo**: Assim, no nosso computador, teremos que acessar essa chave que foi gerada com o SSH.

No terminal integrado, quando pressionamos "yes", ele deve ter criado uma chave que fica salva dentro do computador na pasta do usuário. Podemos tanto fazer isso pelo terminal ou navegar até a pasta, o que é mais fácil.

Portanto, vamos abrir o explorador de arquivos e entrar na pasta do usuário. Nesse casso, a pasta Rodrigo. Agora, vamos procurar uma pasta oculta chamada .ssh. Entrando nela, há um arquivo chamado known\_hosts, que são os hosts conhecidos, mas a chave não foi gerada.

Para gerar a chave, existe um comando que temos que copiar. Inclusive, existe um tutorial no próprio site do GitHub, que vamos disponibilizar no curso como uma atividade. Ele tem um tutorial ensinando [como gerar uma chave SSH](https://docs.github.com/pt/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent) para cada sistema operacional. No nosso caso, é o Windows.

Nessa documentação, temos a explicação do que é e como gerar uma chave SSH. Vamos copiar o comando sugerido ssh-keygen -t ed25519 -C seguido do e-mail da conta do GitHub.

No terminal do Visual Studio Code, vamos colá-lo. Porém, temos que alterar o e-mail de exemplo para o nosso e-mail, que é rodrigo.alura87@gmail.com. Você deve substituir com seu e-mail pessoal.

ssh-keygen -t ed25519 -C "rodrigo.alura87@gmail.com"

Copiar código

Após apertar "Enter", ele informa que vai gerar um par de chaves SSH. E nos pergunta onde queremos salvar esse arquivo.

Enter file in which to save the key (C:\User\Rodrigo/.ssh/id\_ed25519):

Você pode simplesmente pressionar "Enter" para salvá-lo na própria pasta SSH, que é a pasta padrão. Ele pergunta se você quer digitar uma senha.

Enter passphrase (empty for no passphrase):

Se digitamos uma senha, toda vez que eu for sincronizar com o GitHub, ele vai nos pedir essa senha. Por isso, vamos apenas pressionar "Enter" para não ter uma senha. Depois, devemos digitar a mesma senha novamente, damos "Enter" novamente.

Pronto. Agora a chave foi gerada.

Vamos até a pasta "Rodrigo > .ssh". Agora, contém dois arquivo chamamos id\_ed25519, um com a chave privada, outro com a chave pública. Devemos abrir o da chave pública, vamos conferir.

Vamos dar dois cliques e mandar abrir com o bloco de notas. Estava escrito *private key*, era a chave privada.

Vamos abrir o outro arquivo. Podemos clicar com o botão direito e selecionar "Abrir com" e escolher o aplicativo de bloco de notas. Agora, temos o código da chave começando com ssh ed-25519 que precisamos copiar e colar no site do GitHub.

No campo *key*, vamos colá-lo e apertar o botão "*Add SSH Key*" para adicionar a chave SSH. O GitHub já adicionou e listou a chave do notebook pessoal em "SSH keys".

Agora o GitHub sabe que neste notebook tem uma chave e já nos autenticamos, pois geramos essa chave com o nosso e-mail. Assim, o GitHub confia neste computador. Desse modo, conseguiremos sincronizar com a nossa conta no GitHub.

**Sincronizar repositórios**

**Rodrigo**: Vamos voltar para o Visual Studio Code e tentar fazer novamente aquele git push:

git push -u origin **main**

Copiar código

Vamos conferir se agora vai funcionar.

Total 14 (delta 0), reused 0 (delta 0), pack-reused 0

To github.com: rodrigoalura87/numero-secreto.git

* [new branch) main -> main

branch 'main' set up to track 'origin/main'.

Finalmente, o upload dos arquivos e do *commit* para o nosso repositório no GitHub foi bem-sucedido, ao que tudo indica. Será que está sincronizado, Gabi?

**Gabrielle**: Sim, aparentemente funcionou, mas vamos abrir nosso repositório no navegador e atualizar a página para verificar se encontramos nosso projeto lá?

**Rodrigo**: No navegador, precisamos voltar para a página do repositório. Basta clicar no ícone do avatar e acessar "*Your repositories*" (seus repositórios) e escolher o numero-secreto. Em seguida, vamos atualizar a página (atalho "F5") e o projeto apareceu.

O tutorial que estava antes sumiu, porque agora tudo já está sincronizado e os arquivos estão sincronizados no repositório remoto.

Temos o index.html, o app.js e o sistema identificou o *commit* e usuário, o rodrigoalura87. Podemos navegar pelos arquivos e ter acesso ao código-fonte.

Logo, nosso repositório está sincronizado e na barra de endereços temos a URL do repositório. No nosso caso, é github.com/rodrigoalura87/numero-secreto.

Podemos compartilhar esse link com outras pessoas e elas podem acessar o repositório, inclusive você, Gabi. Eu posso te enviar este repositório e fica aí essa pergunta: será que você conseguirá baixar o projeto no seu computador?

**Gabrielle**: O GitHub nos proporciona essa possibilidade de outras pessoas não apenas acessarem nosso código, mas também contribuírem para ele. Vamos explorar isso em seguida.

**12Para saber mais: Chave SSH**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/150170/next)

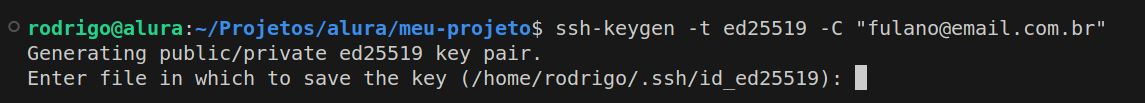
O comando git push deve ser executado para **sincronizar** as mudanças do repositório local com o repositório remoto, ou seja, quando desejamos enviar os novos commits que realizamos em nosso repositório local para o repositório remoto. No entanto, para garantir uma conexão segura, é essencial configurar uma **chave SSH** no computador antes de executar esse comando.

**Chave SSH**

Ao vincular um repositório remoto ao nosso repositório local, via comando git remote add, precisamos utilizar algum protocolo seguro, como HTTPS ou SSH. No caso de se utilizar o protocolo SSH, escolha realizada neste curso, devemos gerar uma chave SSH em nosso computador, além de cadastrá-la em nossa conta do GitHub. Isso é necessário para garantir a autenticação, pois o GitHub checa se quem está realizando o push dos commits tem permissão para realizar tal ação.

**Geração de uma chave SSH**

Antes de executar o comando git push, precisamos gerar uma chave SSH. A geração da chave é feita via terminal, com o comando ssh-keygen -t ed25519 -C "SEU EMAIL AQUI":



Repare, na imagem anterior, que ao executar o comando para gerar uma chave SSH, uma pergunta foi feita e o terminal fica travado esperando nossa resposta:

Generating **public**/**private** ed25519 **key** pair.

Enter file **in** which **to** save the **key** (/home/rodrigo/.ssh/id\_ed25519):

Copiar código

Essa primeira pergunta é para indicarmos o **diretório** em nosso computador no qual a chave será salva, sendo que entre parênteses é indicado o diretório padrão. O recomendado é apenas apertar a tecla enter no teclado para que a chave seja salva no diretório padrão, pois assim o Git consegue encontrar essa chave automaticamente sempre que executarmos o comando git push.

Após apertar a tecla enter, uma nova pergunta será apresentada no terminal:

Enter **passphrase** (enter **for** no passphrase):

Copiar código

Essa segunda pergunta é para indicarmos se desejamos adicionar uma senha à chave SSH que será gerada. Caso você digite uma senha, toda vez que executar o comando git push será necessário digitar tal senha. Ao não digitar nada e apenas apertar a tecla enter, a chave será gerada sem a proteção de uma senha.

Por fim, a terceira e última pergunta é apenas para confirmar a senha anterior:

Enter same passphrase again:

Copiar código

A chave será gerada e a seguinte mensagem será exibida no terminal:

Your identification has been saved **in** /home/rodrigo/.ssh/id\_ed25519

Your **public** **key** has been saved **in** /home/rodrigo/.ssh/id\_ed25519.pub

The **key** fingerprint is:

SHA256:jxAkhGR7NHm/0fcmyPnErZxSKr+ObsH7r4AC/vUNvPY fulano@email.com.br

The **key**'s randomart image is:

+--[ED25519 256]--+

| .oo=.. |

| ..o.+. |

| . .... . |

| . .o . . |

| . ..S+ = o |

| . . ++o+ = + |

| . . o =o.\* = |

| . o .=\*o = |

| . +=\*E=. |

+----[SHA256]-----+

Copiar código

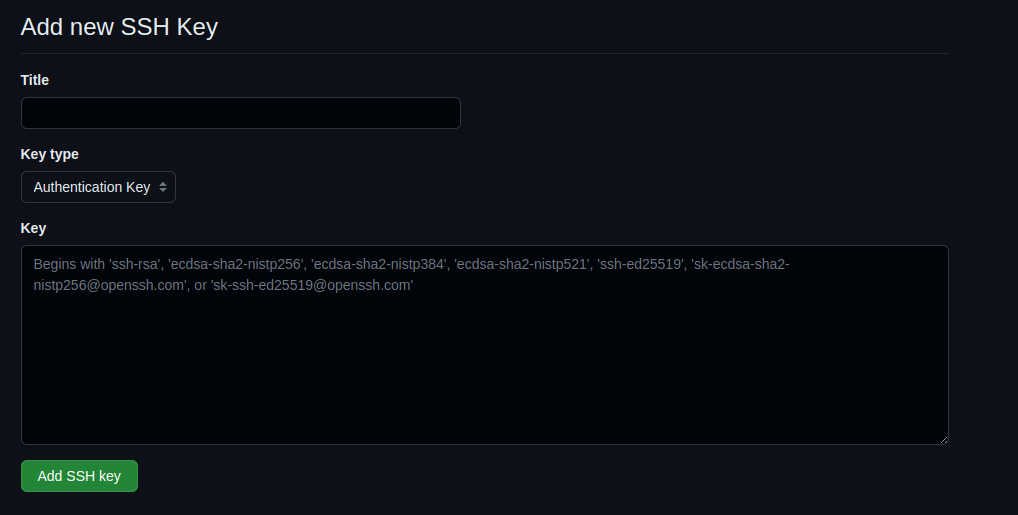
Na primeira linha da mensagem você consegue identificar o diretório no seu computador no qual a chave foi salva. Agora, basta acessar tal diretório para ter acesso à chave SSH.

**Observação:** Nesse diretório serão gerados dois arquivos que representam a chave SSH, sendo um para a chave privada (arquivo id\_ed25519) e o outro para a chave pública (id\_ed25519.pub).

**Cadastrando a chave SSH no GitHub**

Após gerar a chave, precisamos cadastrá-la em nossa conta do GitHub, para que assim o GitHub consiga nos identificar e autenticar ao executar o comando git push de nosso computador.

Acesse a [página de chaves SSH](https://github.com/settings/keys) de sua conta no GitHub e clique no botão **New SSH key** ou **Nova chave SSH** para realizar o cadastro da chave:



Repare que o formulário exibido na imagem anterior contém três campos:

* **Title** ou **Título**: Informe um apelido para sua chave SSH (por exemplo: computador casa)
* **Key type** ou **Tipo de chave**: Escolha o tipo **Authentication Key** ou **Chave de autenticação**
* **Key** ou **Chave**: Nesse campo você deve colar o **conteúdo** do arquivo da sua chave SSH **pública** (arquivo id\_ed25519.pub)

Após realizar esse procedimento, será possível sincronizar o repositório local com o remoto, enviando os novos commits com o comando git push.

**13Para saber mais: O comando git remote**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/141645/next)

Vimos no vídeo anterior que para fazer o link entre o repositório local, que está em nosso computador, com o repositório remoto, que está no GitHub, devemos utilizar o comando git remote.

Esse comando tem algumas variações e parâmetros opcionais que podem ser úteis em certas situações. Confira a seguir exemplos de uso desse comando:

1 - Adicionar um repositório remoto:

Quando você deseja estabelecer uma conexão entre seu repositório local e um repositório remoto, como aquele hospedado no GitHub, o comando git remote add é a ferramenta essencial. Essa etapa é crucial para possibilitar a colaboração e o compartilhamento de código com outras pessoas, bem como para fazer backup de seu trabalho em um servidor remoto.

A sintaxe básica do comando é a seguinte:

git remote **add** apelido url

Copiar código

'apelido': Este é um nome que você atribui ao repositório remoto. Geralmente, se utiliza nomes descritivos, como "origin" para o repositório principal no GitHub, mas você pode escolher nomes que façam sentido para o seu projeto.

'url': Aqui, você insere a URL do repositório remoto. Esta URL é única para cada repositório remoto e serve como o endereço para acessar e interagir com ele pela internet. Certifique-se de copiar a URL correta do repositório que deseja adicionar.

2 - Listar os repositórios remotos:

Para listar os repositórios remotos associados ao seu projeto local, você pode utilizar o comando git remote -v. Isso exibirá uma lista de todos os repositórios remotos vinculados ao seu projeto, juntamente com suas URLs. Veja um exemplo:

git remote -v

Copiar código

A saída será algo semelhante a:

origin https://github.com/seu-usuario/seu-projeto.git (fetch)

origin https://github.com/seu-usuario/seu-projeto.git (**push**)

Copiar código

Essa lista é útil para verificar se os repositórios remotos estão configurados corretamente. Obs: Vai aparecer duplicado mesmo, pois o Git separa a url de envio de commits (push) da url de baixar commits (fetch).

3 - Remover um repositório remoto:

Caso você não precise mais de um repositório remoto específico, pode removê-lo com o comando git remote remove apelido. Substitua 'apelido' pelo nome do repositório remoto que deseja remover. Aqui está um exemplo:

git remote **remove** origin

Copiar código

Após a execução deste comando, o repositório remoto chamado "origin" será desvinculado do seu projeto local.

4 - Alterar a URL de um repositório remoto:

Às vezes, é necessário atualizar a URL de um repositório remoto, como quando a URL do servidor do GitHub é modificada ou quando você copiou a URL incorreta ao adicionar o repositório remoto. Use o comando git remote set-url apelido nova\_url para realizar essa atualização. Substitua 'apelido' pelo nome do repositório remoto e 'nova\_url' pela nova URL que você deseja associar a ele. Aqui está um exemplo:

git remote set-url origin https://github.com/seu-usuario/seu-repositorio.git

Copiar código

Isso atualizará a URL do repositório remoto "origin" para a nova URL especificada.

5 - Alterar o apelido de um repositório remoto:

Se você deseja renomear um repositório remoto, use o comando git remote rename apelido novo\_apelido. Substitua 'apelido' pelo nome atual do repositório remoto e 'novo\_apelido' pelo novo nome que você deseja atribuir a ele. Aqui está um exemplo:

git remote rename origin novo-origin

Copiar código

Isso renomeará o repositório remoto de "origin" para "novo-origin".

Lembre-se de que o comando git remote é fundamental para a gestão de conexões entre seu repositório local e repositórios remotos, permitindo a colaboração eficiente e o controle de versão. Praticar esses comandos em seu ambiente de desenvolvimento ajudará a consolidar seu entendimento.

**Opinião do instrutor**

Você precisará criar um repositório no GitHub e executar os comandos do git no terminal do projeto em seu computador:

git init

git **add** .

git **commit** -m "projeto inicial"

git remote **add** origin git@github.com:SEU\_USUARIO/SEU\_REPOSITORIO

git push -u origin main

Copiar código

Lembre-se de substituir SEU\_USUARIO pelo seu username definido ao criar a sua conta no GitHub e SEU\_REPOSITORIO pelo nome do repositório que você criou no GitHub.

**16Para saber mais: autenticação por token no Github**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139730/next)

Nesse curso, aprendemos como realizar a autenticação no Github em seu computador pessoal através de uma chave SSH. Entretanto, essa não é a única forma de se autenticar no Github.

Caso você tente conectar seu repositório local com o repositório remoto através da conexão HTTP, será pedido uma senha no momento de realizar o seu login. Porém, desde 2021, por motivos de segurança, o Github não aceita mais que essa autenticação seja feita por meio de sua senha.

Caso você tente usar sua senha, acontecerá um erro como o mostrado a seguir:

Logon failed, use ctrl+c to cancel basic credential prompt.

remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.

remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.

fatal: unable to access ‘<repositório Git>’: The request URL returned error: 403

Copiar código

Assim, a outra opção de autenticação para usar o Github em seu terminal é através de um **token para o git**.

O artigo [Nova exigência do Git de autenticação por token, o que é e o que devo fazer?](https://www.alura.com.br/artigos/nova-exigencia-do-git-de-autenticacao-por-token-o-que-e-o-que-devo-fazer) te ensinará a realizar as configurações necessárias para criar e utilizar esse token.

**17Desafio: hora da prática.**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/145912/next)

O Git é uma ferramenta excelente para acompanhar a mudança entre versões de um mesmo projeto e, dentre vários benefícios, nos ajuda a observar de perto o desenvolvimento do seu aprendizado. Tudo isso de uma forma organizada através dos commits.

Além disso, algo que é essencial no universo da tecnologia é apresentar o seu progresso para a comunidade e montar um portfólio dos seus projetos para demonstrar suas habilidades. Dessa forma, o GitHub é essencial para compartilhar e colaborar em projetos de programação de todo o mundo.

Pensando nisso, criamos uma lista de atividades (não obrigatórias) focada em prática para melhorar ainda mais sua experiência de aprendizagem. Bora praticar então?

**Desafios**

1. Crie uma conta no GitHub
2. Crie um repositório para um projeto pessoal.
3. Faça a instalação do Git
4. Crie um repositório localmente para o seu projeto pessoal
5. Adicione alguns arquivos no seu repositório local
6. Faça o commit das alterações
7. Configure a identidade do autor do commit.
8. Crie a branch Main
9. Realize a conexão do seu repositório local com o remoto
10. Envie as alterações no repositório local para o remoto
11. Utilize o comando git status

Caso precise de ajuda, opções de solução das atividades estarão disponíveis na seção “Opinião da pessoa instrutora”.

**Opinião do instrutor**

1. Crie uma conta no GitHub Você pode começar seguindo os passos mostrados no vídeo, preenchendo o formulário de cadastro, verificando sua conta e explorando a página inicial do GitHub.
2. Criar um novo repositório no GitHub e fazer o upload de um projeto local para esse repositório. Você pode seguir passos mencionados na aula, como acessar as configurações do GitHub, criar um novo repositório com um nome único e escolher se ele será público ou privado. Em seguida, você pode adicionar uma descrição, um README, um .gitignore e uma licença ao repositório.
3. Instalação do Git: Caso você ainda não tenha realizado a instalação, siga os passos na atividade [Faça como eu fiz: instalação do Git](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/144999)
4. Para criar um repositório local você, digite o seguinte comando no terminal: git init
5. Para adicionar os arquivos no repositório local, digite no terminal o comando: git add .
6. Faça um commit com as modificações, digite no terminal o comando: git commit -m "mensagem de commit"
7. Para configurar a identidade do autor do commit, digite no terminal o comando:
8. git -config --**global** user.email "seuemailaqui@example.com"
9. git config --**global** user.name "seu nome aqui"

Copiar código

1. Para criar a branch Main, digite no terminal o comando: git branch -M main

O comando git branch -m é usado para criar uma nova ramificação no repositório Git atual. Neste caso, criamos a branch padrão main, que representa a versão principal do código.

1. Para realizar a conexão do seu repositório local com o remoto via SSH, digite no terminal: git remote add origin git@github.com:seunomedeusuario/seu-repositorio.git

Caso seja necessário, realize a configuração do protocolo SSH através da geração de chave, você pode acompanhar os passos em vídeo na atividade [Sincronizando repositórios](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139310)

1. Para subir as alterações no repositório local para o remoto, digite o seguinte comando no terminal: git push -u origin main
2. Digite no terminal o comando git status e observe a saída.

O comando git status é uma ferramenta essencial para gerenciar alterações no controle de versão Git. Ele fornece uma visão geral do estado atual do repositório, indicando quais arquivos foram modificados, adicionados ou excluídos desde o último commit. Essa informação é crucial para compreender o progresso do desenvolvimento e tomar decisões de gerenciamento de alterações.

A saída do comando git status geralmente contém três seções principais:

**Modificados:** Lista os arquivos que foram modificados desde o último commit, mas ainda não foram adicionados à área de preparação (Stage).

**Adicionados:** Indica os arquivos que foram adicionados à área de preparação, mas ainda não foram confirmados no histórico de commits.

**Modificados, adicionados ou excluídos:** Exibe os arquivos que não foram rastreados pelo Git, ou seja, que não foram adicionados ao índice de modificações (Staging Area).

Além disso, o comando git status pode fornecer informações adicionais sobre as ramificações atuais, como a ramificação atual e as ramificações que estão à frente ou atrás da atual.

O comando git status é uma ferramenta indispensável para qualquer desenvolvedor que utiliza Git. Ele permite monitorar as alterações no repositório, identificar o estado dos arquivos e tomar decisões de gerenciamento de commits de forma eficiente.

**18O que aprendemos?**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139731/next)

**Nessa aula, você aprendeu como:**

* Criar sua conta no GitHub para poder compartilhar seus projetos de software e/ou colaborar em projetos de outras pessoas;
* Baixar e instalar o Git em seu computador;
* Configurar sua conta do Git em seu computador, com os comandos git config --global user.name e git config --global user.email;
* Criar um repositório remoto no GitHub;
* Criar e conectar seu repositório local com o repositório remoto por meio dos comandos git init, git add, git commit, git remote add e git push.

Transcrição

Rodrigo: Concluímos a primeira etapa do projeto! Criamos uma conta no GitHub, instalamos o Git localmente no computador, criamos o repositório remoto e o local e fizemos a ligação entre eles. Além disso, configuramos a chave SSH e fizemos as configurações iniciais.

Importante lembrar que, nos vídeos anteriores mostramos a chave privada e a pública apenas para demonstrar como funcionam. Na prática, você não compartilhará essas informações, pois elas são restritas ao seu computador.

Agora daremos continuidade. O repositório remoto no GitHub já possui os arquivos, então, compartilhei o projeto com você, Gabi. Assim, você poderá baixar o código e importar no seu VS Code para fazer modificações no projeto.

Gabrielle: Ótimo, Rodrigo! Com o navegador aberto, já consigo acessar e visualizar todos os arquivos do projeto. Agora vamos transferir esse repositório para o meu computador.

Na lateral direita da tela, clicamos no botão verde nomeado "Code". Feito isso, a ferramenta abre opções para trazer o projeto para o computador. Sendo o Download ZIP, abrir o GitHub no Desktop e clonar o repositório.

Nesse caso, qual é a melhor opção, Rodrigo?

Rodrigo: A princípio realizar o Download ZIP poderia ser uma boa opção, pois o GitHub reúne todos os arquivos do repositório, cria o arquivo zip e faz o download para o seu computador. Depois, você pode descompactar no desktop ou em alguma pasta.

Porém, como nesse caso é realizado apenas o download dos arquivos e não uma cópia do repositório com o commit e todo histórico do projeto. Então, vamos escolher a opção de clonar.

Clonando um repositório

Gabrielle: Na opção de clonar, o GitHub fornece a URL do repositório. Sendo assim, a copiamos e abrimos o explorador de arquivos do computador e acessamos a área de trabalho.

Em seguida, clicamos com o botão direito do mouse e selecionamos "Abrir no terminal". Em seguida, para clonar o repositório, vamos passar o comando git clone seguido da URL que copiamos e apertamos "Enter".

git clone https://github.com/rodrigoalura87/numero-secreto.git

Copiar código

Se voltarmos ao explorador de arquivos, encontramos a pasta do projeto. Ao acessá-la encontramos todos os arquivos do GitHub.

Sabendo disso, abrimos o VS Code. Para acessar a pasta que clonamos, na barra de menu superior, clicamos em "File > Open Folder". Na janela que abre, procuramos pela pasta "numero-secreto" que está na área de trabalho, selecionamos e clicamos em "Selecionar pasta". Assim o VS Code abre o projeto.

Rodrigo: Ótimo! Então, mesmo que esteja na minha conta do GitHub, você consegue baixar o projeto. Isso porque quando criamos o repositório o deixamos como público, ou seja, qualquer pessoa pode acessá-lo do navegador, pode visualizar os arquivos, fazer o download ou cloná-lo.

Gabi, agora você tem o projeto no seu computador e você tem acesso ao código-fonte. Será que você consegue modificar os arquivos, fazer commits e mudanças no projeto?

Gabrielle: Sim, isso é possível. Descobriremos como no vídeo seguinte. Te esperamos lá!

**Transcrição**

**Rodrigo:** Gabi, você já conseguiu baixar o projeto do repositório público da minha conta do GitHub e também importou na IDE, no Visual Studio Code.

Agora, vamos descobrir se você consegue trabalhar no mesmo projeto que desenvolvi e fiz upload para o repositório no GitHub.

O que você acha de aumentarmos a dificuldade do jorgo Número Secreto? Atualmente, ele sorteia um número de 1 a 10. Que tal modificarmos para escolher um número aleatório entre 1 e 100?

**Gabrielle:** Essa é uma alteração interessante, vamos lá!

**Realizando um commit**

index.hmtl

Começamos abrindo o arquivo index.html. Feito isso, na linha 23, no texto Escolha um número entre 1 a 10, mudamos para 10 a 100.

//Código omitido

<p **class**="texto\_\_paragrafo">Escolha um número entre 1 a 100</p>

//Código omitido

Copiar código

app.js

Em seguida, abrimos o arquivo app.js. Na linha 2, mudamos a variável numeroLimite para 100.

**let** numeroSecreto = parseInt(**Math**.**random**() \* 11)

**let** tentativas = 1

**let** chute

//código omitido

Copiar código

**Rodrigo:** Estamos simulando uma mudança no código do projeto, como se tivesse sido solicitada por um usuário ou cliente.

Sendo assim, realizamos as alterações, porém ainda não estão registradas no repositório local do computador.

**Gabrielle:** Isso mesmo, precisamos registrar essas alterações. Para isso, na barra de menu superior do VS Code, clicamos em "Terminal" e depois em "New terminal" para abrí-lo.

**Rodrigo:** Nessa funcionalidade você precisou mexer em dois arquivos, certo? Suponhamos que isso foi feito em uma sexta-feira e quando você voltou ao trabalho na semana seguinte, não lembra onde parou no projeto.

Tem algum comando no GitHub para sabermos o estado atual do código?

**Gabrielle:** Sim, o comando é o git status.

git status

Copiar código

Ao executá-lo é exibido na tela informações sobre as alterações feitas nos arquivos app.js e index.html do projeto e que essas mudanças que precisam ser registradas no commit.

**Rodrigo:** Então, se esquecermos qual arquivo foi modificado ou qual precisa ser adicionado, basta executar esse comando que será exibido o status atual do repositório local.

No nosso caso, temos dois arquivos modificados que estão destacados em vermelho porque ainda não foram adicionados para fazer o commit na sequência.

**Gabrielle:** Repare que o próprio git status nos fornece uma dica para usar o comando git add seguido do nome do arquivo para fazer essa adição.

Anteriormente descobrimos que temos a opção de adicionar arquivo por arquivo, escrevendo o nome de cada um, ou podemos adicionar todas as mudanças de uma vez com o ponto. Portanto, usaremos o git add ..

git **add** .

Copiar código

Feito isso, provavelmente os arquivos foram adicionados Para conferir, rodamos novamente o git status.

git status

Copiar código

Repare que ao fazer isso, notamos no terminal que a ferramenta indica que existem alterações para serem comitadas e quais foram os arquivos modificados. Além disso, a cor mudou para verde, isso significa que os arquivos foram adicionados corretamente.

**Rodrigo:** Muito legal, Gabi. O GitHub tem essa sinalização pelas cores, se os arquivos modificados, mas não foram adicionados ficam vermelhos quando são ficam verdes.

Agora que você já adicionou, podemos fazer o commit, que é o comando responsável por registrar uma modificação no código.

**Gabrielle:** Para isso, passamos o comando git commit. Lembrando que todo commit sempre terá uma mensagem que informará qual alteração foi feita no projeto.

Para adicionar essa mensagem, na mesma linha de comando escrevemos -m "alterando limite para 100" e apertamos "Enter".

git **commit** -m "alterando limite para 100"

Copiar código

Como retorno é informado que o commit foi realizado com sucesso.

**Rodrigo:** Importante ressaltar que, quando fazemos um commit é como se estivéssemos registrando uma versão do nosso sistema. Se estávamos na versão 1, passamos para a versão 2, depois para a versão 3, 4, 5, e assim por diante.

Porém, precisamos saber o que é a versão 1, o que mudou na versão 2, 3 e 4. É justamente a mensagem que proporciona esse indicativo.

Disponibilizamos no Para Saber Mais dicas de **boas práticas** para desenvolver essa mensagem para que fique compreensível e as pessoas entendam as mudanças e o que cada alteração muda no projeto.

Inclusive, Gabi, existe algum comando em que conseguimos identificar quais commits foram realizados no projeto e quem fez?

**Gabrielle:** Tem, sim, Rodrigo! Para isso, podemos utilizar o comando git log.

git log

Copiar código

Ao executá-lo é exibido o histórico dos commits que foram feitos ao longo do projeto.

Se analisarmos o retorno, retonhamos o primeiro commit que o Rodrigo fez para subir o projeto inicial. Identificamos o nome do autor, data seguido da mensagem. Logo acima temos o commit mais atual que é o que acabamos de fazer.

**Rodrigo:** Observe que o git log mostra o log, ou seja, o histórico de commits realizados no projeto. Assim, podemos identificar quem foi que fez cada registro, a mensagem, o significado de cada registro, a data e toda essa sequência de alterações no repositório.

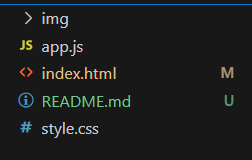
Ótimo! Gabi, você conseguiu fazer uma modificação no projeto e efetuou o commit. Porém, é importante lembrar que o commit está apenas no seu repositório local, ainda não foi enviado para o *GitHub*.

Na sequência descobriremos como enviar esse commit para o repositório no *GitHub*.

**03Para saber mais: sinalizações em arquivos do VSCode**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/143209/next)

Quando estamos trabalhando em um projeto utilizando o versionamento Git e a IDE VSCode, ao adicionar ou alterar algum arquivo aparece uma sinalização ao lado do nome desses arquivos no VSCode, como podemos ver na imagem abaixo:



Mas o que isso significa?

* **M**: A letra M representa o estado *Modified*, do português modificado. Isso significa que o arquivo já existia no repositório, mas que recebeu alguma modificação que ainda não foi registrada no Git.
* **U**: A letra U representa o estado *Untracked*, do português não rastreado. Isso significa que o arquivo ainda não existia no repositório e que ainda não teve seu registro (commit) feito no Git.

Essa sinalização nos ajuda a entender o estado atual dos nossos arquivos do projeto no versionamento Git.

**04Para saber mais: mensagens de commits**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139732/next)

Apesar de não existir uma regra universal para a escrita das mensagens de commit, algumas boas práticas podem ser seguidas para garantir que outras pessoas, e até mesmo você no futuro, entendam que alterações foram feitas e por quê.

As mensagens dos commits devem ser **simples e objetivas**. A seguir, listamos algumas orientações para isso:

* **Mantenha a mensagem curta e concisa:** A primeira linha da mensagem deve conter, no máximo, 72 caracteres. Caso seja necessário uma descrição adicional, você deve pular uma linha e adicionar os detalhes, como o contexto, da mudança realizada.
* **Uso de verbo no infinitivo:** É comum que a mensagem do commit inicie com um verbo no infinitivo que descreva a alteração feita, como “adicionar”, “corrigir” ou “atualizar”. Em sequência, são adicionados detalhes concisos da mudança. Por exemplo: “Atualizar texto do título da página”.
* **Evite detalhes técnicos**: Não inclua detalhes técnicos complexos na mensagem de commit. Esses detalhes podem ser adicionados nos comentários de código ou na documentação.

É importante ter em mente que a mensagem do commit é uma forma de documentação do histórico das mudanças que ocorreram ao longo do código. A pessoa que ler essa mensagem pode não ter conhecimento do contexto original. Assim, garanta que suas mensagens de commit tenham clareza e sejam suficientemente descritivas.

**05Para saber mais: quando realizar um commit?**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139733/next)

Um commit deve ser realizado sempre que você **finalizar uma tarefa** específica ou **resolver algum bug**. Isso mantém o histórico de commits claro e rastreável, de modo que seja possível entender o que foi feito em cada commit.

Assim, é importante realizar commits frequentemente. Porém, evite realizar commits muito pequenos ou muito grandes, pois isso pode tornar difícil o seu entendimento.

Lembre-se de **nunca realizar um commit de um código que você sabe que contém bugs**. O ideal é que o commit contenha somente código funcional.

**06Para saber mais: como o Git controla as mudanças?**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139734/next)

O controle de mudanças do Git é feito através dos **commits**. Cada commit armazena o **estado completo** do projeto em um determinado momento por meio de um **snapshot**. Ou seja, cada commit é um registro completo do repositório no momento em que esse commit foi criado.

Como cada commit é uma representação completa e consistente do estado do projeto em um determinado ponto no tempo, isso facilita a rastreabilidade e o entendimento de como se deu a evolução do código ao longo do tempo.

Todo commit conta com um id único e traz uma referência aos commits anteriores. Assim, através dessa cadeia de commits, o Git registra um **histórico completo** de todos os commits realizados no repositório.

Caso queira conhecer melhor sobre esse processo, acesse a [documentação oficial do Git](https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-O-B%C3%A1sico-do-Git).

**Transcrição**

**Rodrigo:** Gabi, você conseguiu clonar o repositório para o seu computador local, fazer uma alteração no código e o commit.

Porém, essa alteração só existe no seu repositório local. Então, tentaremos fazer o mesmo processo realizado anteriormente, enviaremos o commit para o GitHub utilizando o comando git push.

**Gabrielle:** Mas, nesse caso, precisamos repetir a conexão com o repositório?

**Rodrigo:** Anteriormente, quando seguimos o tutorial, primeiro utilizamos o comando git remote add para adicionar no repositório local um link com o repositório remoto. Só depois fizemos o git push. Porém, você não precisará repetir esses passos, pois rodou o comando clone do repositório.

**Gabrielle:** O comando clone já realiza automaticamente essa conexão entre o repositório remoto e o repositório local.

**Rodrigo:** Inclusive, você pode verificar isso rodando o comando git remote sem passar nenhum parâmetro. Ao fazer isso, será listado os repositórios remotos no seu repositório local.

**Gabrielle:** Então, vamos passar o comando git remote seguido de "Enter".

git remote

Copiar código

Feito isso é exibido o origin.

**Rodrigo:** O origin é o nome que o GitHub atribui quando fazemos o clone, é como se referisse a origem do repositório.

Como você já tem o repositório remoto adicionado, agora pode executar o comando git push para tentar enviar esse commit para o meu repositório no GitHub.

**Gabrielle:** Passamos git push e em sequência precisamos informar para onde enviaremos o commit. Escrevemos origin que é o apelido do repositório remoto, seguido de main que é a branch e apertamos "Enter".

Embora não vamos nos aprofundar nesses termos agora, em breve você encontrará um material explicativo para aprofundar ainda mais seu conhecimento sobre o tema.

git push origin **main**

Copiar código

Ao executar o comando notamos um erro.

**Rodrigo:** É um erro de permissão, certo? Já esperava que isso acontecesse, afinal, se o erro não aparecesse teria algo errado.

O repositório foi criado na minha conta do GitHub com meu e-mail pessoal e você tem outra conta. Embora você tenha conseguido baixar o repositório, criado como público, esse é um material público apenas para leitura.

Isso significa que as pessoas podem acessá-lo, baixar o código, fazer mudanças e commits, porém apenas localmente. Quando alguém tentar fazer um push, o próprio GitHub irá barrar.

Isso porque você é outra pessoa usuária que está tentando fazer um push para um repositório da minha conta, Rodrigo. Isso não é permitido, a não ser que eu dê permissão.

**Adicionando colaboradores ao projeto**

**Rodrigo:** Se quisermos que outras pessoas colaborem nesse projeto, é preciso adicioná-las manualmente no projeto. Para isso, no navegador, logamos no Github e acessamos a página do repositório numero-secreto.

Na barra de menu superior, clicamos no botão "Settings", que se refere a configuração. Em seguida, somos encaminhados para uma nova tela.

No menu lateral esquerdo, clicamos em "Collaborators". Feito isso, notamos que não há nenhum colaborador. Para adicionarmos, no fim da tela, clicamos no botão verde chamado "Add people".

Na nova janela, há um campo referente ao username da pessoa que adicionaremos como colaboradora. Preenchemos com o user da Gabi e depois clicamos no botão "Add".

Ao fazer isso, a ferramenta indica que o pedido de acesso está pendente, pois a Gabi precisa aceitá-lo. Sendo assim, esse processo não é automático.

Então, Gabi, esse é seu momento de compartilhar a tela.

**Gabrielle:** No navegador, acesso meu e-mail do GitHub. Feito isso, notamos o e-mail do GitHub com o convite. Para visualizá-lo, clicamos no botão "View invitation".

**Rodrigo:** Repare que, se voltarmos a minha tela, a Gabi agora aprece como colaboradora do repositório.

Isso é o que você fará no dia a dia de trabalho. Quando estiver em um repositório e quiser que outras pessoas colaborem, será necessário adicioná-las como colaboradoras.

Gabi, agora, se você tentar rodar o comando push é para funcionar. Vamos testar?

**Gabrielle:** Vamos nessa. No terminal do VS Code, digitamos novamente o comando git push origin-main e pressionamos *Enter*.

git push origin-**main**

Copiar código

Feito isso, é exibido uma mensagem que o commit foi enviado para o repositório.

**Rodrigo:** Vamos conferir. No site do GitHub, na barra de menu superior, clicamos em "Code" para voltarmos para a página inicial do repositório.

No arquivo app.js conseguimos notar a mudança de "alterando o limite para 100" e ao lado a foto da Gabi, que fez a alteração.

Na lateral superior direita, a ferramenta indica que há dois commits. Se clicarmos, visualizamos o primeiro que é o projeto inicial e o segundo o atualizado.

Se clicarmos no commit encontramos os arquivos modificados e as alterações realizadas em cada um deles. Então, deu certo, Gabi. Você conseguiu colaborar comigo neste projeto!

**Gabrielle:** Ótimo, Rodrigo! Porém, agora que enviei as alterações, imagino que você queira continuar trabalhando no seu projeto. Como podemos trazer essas modificações que estão no GitHub para o seu computador?

**Rodrigo:** Isso é muito importante! O último commit existe no seu repositório local, porque você o fez no seu computador.

Embora você tenha feito o push, que está no GitHub, no meu repositório local ele ainda não existe, pois não é sincronizado automaticamente.

No vídeo seguinte vamos descobrir como baixar os novos commits para o repositório local.

**Até lá!**

**09Para saber mais: adicionando colaboradores no commit**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139735/next)

Cada commit possui por padrão um autor, que é a pessoa que realizou aquelas alterações no código.

Entretanto, quando trabalhamos em equipe pode ser que algum trecho de código seja escrito em dupla ou trio. Assim, como definir a autoria dessas outras pessoas no commit?

O Git oferece a possibilidade de adicionar **mais de um autor** a um commit. Para isso, após escrever a mensagem do commit, pulamos duas linhas e usamos a palavra-chave Co-authored-by:, seguido do nome e e-mail associado ao GitHub (entre < >) de cada pessoa colaboradora.

Cada coautor deve estar em uma linha diferente, como é mostrado no exemplo a seguir:

$ git commit -m "Adicionar nova funcionalidade.

>

>

Co-authored-by: NOME <nome@email.com>

Co-authored-by: OUTRO-NOME <outro@email.com>"

Copiar código

Caso queira entender mais sobre coautoria no GitHub, você pode acessar a [documentação referente ao assunto](https://docs.github.com/pt/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/creating-a-commit-with-multiple-authors).

**10Para saber mais: outras formas de colaborar**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139736/next)

Existem diversos projetos de software com seu código fonte disponível no GitHub e abertos para colaboração de qualquer um que queira contribuir. Esse modelo de desenvolvimento é chamado de **Open Source** ou **Código Aberto**.

Caso queira entender mais sobre projetos Open Source, temos aqui na Alura o artigo [Open Source - Uma breve introdução](https://www.alura.com.br/artigos/open-source-uma-breve-introducao), que fala mais sobre o tema.

Em um projeto que segue o modelo Open Source, as demandas, como novos recursos e correção de bugs, são descritas e listadas no repositório do GitHub via **issues**. Assim, caso você queira colaborar, é possível escolher uma issue.

Você precisará realizar um **fork** do projeto, que é uma cópia do repositório em sua conta. Assim, você poderá escrever o código que soluciona a issue escolhida.

Por fim, para enviar sua solução de volta ao repositório fonte, você precisará abrir um **pull request**, que é uma solicitação de pull das suas alterações. Esse pull request passará por um processo de avaliação dos responsáveis pelo projeto, podendo ser aceito ou não.

Caso seja aceito, seu código agora fará parte do código fonte desse projeto.

É importante sempre se atentar às **regras de contribuição** de cada repositório, que podem variar de acordo com o projeto.

Grandes projetos são Open Source e se encontram no GitHub, como a IDE VS Code e o framework React, do JavaScript. Você pode conferir os repositórios desses projetos nos links abaixo:

* [Repositório do VS Code no Github](https://github.com/microsoft/vscode)
* [Repositório do React no Github](https://github.com/facebook/react-native)

Transcrição

Gabrielle: No vídeo anterior, fizemos alterações no projeto e as enviamos. Porém, ainda não estão no repositório local do Rodrigo. Vamos descobrir como fazer isso.

Baixando novos commits

Rodrigo: Vamos lá! No GitHub, já temos o seu commit, mas se abrirmos o VS Code, o código continua na versão antiga, na qual o número limite que pode ser sorteado é 10.

Isso significa que o repositório local não atualizou automaticamente o commit feito pela Gabi. Para isso, existe o comando git pull especificamente para isso. Ele funciona como o oposto do push, já que puxa os commits do remoto para o local.

Para isso, além de git pull, na mesma linha de código indicamos qual é o repositório remoto do qual baixaremos esses commits, nesse caso será o origin. Em seguida, passamos a branch main, onde ele trará esses commits para o repositório local.

git pull origin main

Copiar código

Após dar "Enter", a ferramenta irá sincronizar com o GitHub. Nisso, aparece uma mensagem dizendo que há um commit, porém no remoto há outro, então esse é baixado.

Gabrielle: Para verificarmos se isso foi feito, podemo rodar o git log.

`git log`

Copiar código

Rodrigo: Assim, temos o commit feito pela Gabi, seguido da data e da mensagem. Se no Explorer abrirmos o arquivo app.js, notamos a mudança no código.

Portanto, o comando git pull tem esse objetivo, baixar os novos commits que outras pessoas colaboradoras do seu repositório enviaram para o GitHub. Com isso, temos um fluxo de trabalho.

Gabrielle: Exatamente, Rodrigo. Quando estamos trabalhando com Git, GitHub, utilizamos sempre esses comandos que aprendemos ao longo desta aula. Vamos relembrá-los!

Quando estivermos trabalhando em um projeto e precisarmos realizar mudanças, usaremos o git status para verificar os arquivos modificados.

Adicionaremos essas mudanças com o comando git add, depois, realizaremos um commit com o git commit. Subiremos essas mudanças para o repositório com o git push e eventualmente, conforme outras pessoas forem colaborando com o projeto, traremos essas mudanças novamente para o computador com o git pull.

Rodrigo: Esse é o fluxo de trabalho no dia a dia. Cada pessoa faz o clone do repositório local e centralizam no repositório remoto do GitHub, permitindo a colaboração de todos.

Por isso utilizar o GitHub para trabalhar de maneira colaborativa com projetos de software é muito interessante.

Gabrielle: Até o momento, digitamos esses comandos no terminal, mas acredito que há uma forma mais fácil de realizar isso.

Rodrigo: É verdade! Nesse caso, estamos usando o VS Code como IDE. Geralmente as IDEs já possuem integração com o Git, pois é o principal sistema de controle de versão.

Então, todo trabalho que fizemos aqui no terminal conseguimos fazer dentro do VS Code de uma maneira visual, tornando esse trabalho muito mais simples.

É isso que estudaremos na aula seguinte!

**13Faça como eu fiz: baixando e modificando um projeto**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139738/next)

Agora é com você! Baixe uma cópia do repositório remoto no GitHub para o seu computador, realize alguma modificação no projeto, registre essas alterações no Git e as envie novamente para o repositório no GitHub.

Ver opinião do instrutor

**Opinião do instrutor**

Para baixar o repositório em seu computador, você deve realizar um clone do projeto através do comando:

git clone https://github.com/SEU\_USUARIO/SEU\_REPOSITORIO.git

Copiar código

Lembre-se de substituir o SEU\_USUARIO pelo seu próprio usuário do GitHub e o SEU\_REPOSITORIO pelo nome do seu repositório no GitHub.

Caso você já tenha o projeto em seu computador, ao invés de realizar um clone, você pode baixar somente uma atualização dos commits através do comando:

git pull origin **main**

Copiar código

Depois disso, abra o projeto que você acabou de baixar no VS Code. Altere algo no código, como o limite superior de valores de 10 para 100, nos arquivos de *app.js* e *index.html*.

Após salvar as alterações, visualize quais arquivos foram modificados no repositório local com o comando:

git status

Copiar código

Agora, para registrar as alterações feitas, use os comandos a seguir:

git **add** .

git **commit** -m “MENSAGEM”

Copiar código

Lembre-se de substituir o MENSAGEM, por um texto que descreva brevemente qual alteração você realizou.

Para enviar as alterações feitas para o repositório remoto no GitHub, use o comando a seguir:

git push origin **main**

Copiar código

Caso você queira verificar se funcionou, basta abrir o seu repositório no GitHub, atualizar a página e o novo commit deverá aparecer por lá.

Manipular diferentes versões de um mesmo projeto pode se tornar desafiador. No entanto, o uso eficiente das ferramentas de versionamento elevam a qualidade do código e tornam o fluxo de trabalho muito mais fácil para toda a equipe. Para isso se tornar uma realidade dentro da rotina da pessoa desenvolvedora, é preciso seguir com a prática constante.

Pensando nisso, desenvolvemos uma lista com exercícios não obrigatórios que abrangem desde a criação de um novo repositório remoto até a sincronização com alterações feitas por outros colaboradores. Tudo para que você desenvolva uma compreensão sólida do fluxo de trabalho do Git enquanto explora comandos essenciais.

**Desafios:**

1. Crie um novo repositório remoto no Github e insira um arquivo.
2. Faça um clone do seu repositório remoto para o local.
3. Faça uma nova modificação no repositório remoto.
4. Atualize seu repositório local a partir do Remoto.
5. Utilize o comando git remote -v no terminal.
6. Confira as mudanças nos arquivos.

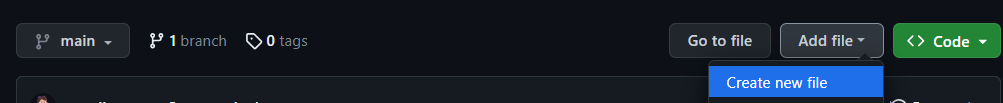
Caso precise de ajuda, opções de solução das atividades estarão disponíveis na seção “Opinião da pessoa instrutora”.

Ver opinião do instrutor

**Opinião do instrutor**

**Desafio 1: Criar um novo repositório remoto no GitHub e inserir um arquivo.**

* Faça login na sua conta do GitHub.
* Clique no sinal de "+" no canto superior direito e escolha "New repository".
* Siga as instruções para criar um novo repositório, dando um nome e uma descrição, se desejar.
* No GitHub, vá para o repositório recém-criado: Clique em "Add file" e escolha a opção que preferir para adicionar um novo arquivo (pode ser um README.md para simplificar).

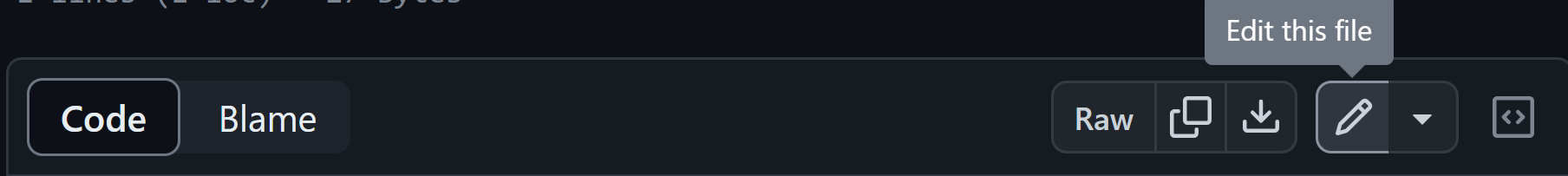
**Desafio 2: Faça um clone do seu repositório remoto para o local.**

* Abra o terminal no seu computador.
* Navegue até o diretório onde deseja clonar o repositório.
* Execute o comando git clone <url-do-seu-repositorio>.

**Desafio 3: Faça uma nova modificação diretamente no GitHub.**

* Acesse o seu repositório no GitHub.
* Abra um arquivo existente ou crie um novo.
* Faça as alterações desejadas diretamente no GitHub e salve.

*Dica: Você pode clicar no arquivo existente e selecionar a opção “Edit this file” (tradução livre para o português: Edite este arquivo)*



**Desafio 4: Atualize seu repositório local a partir do Remoto.**

* Navegue até o diretório do seu repositório local usando o terminal.
* Execute o comando git pull origin main para trazer as últimas alterações do repositório remoto para o seu local.

**Desafio 5: Utilize o comando git remote -v no terminal.**

* Liste repositórios remotos:
* No terminal, navegue até o diretório do seu repositório local.
* Execute git remote -v para listar as entradas remotas configuradas e suas URLs.

O comando é usado para listar todas as entradas remotas configuradas no repositório Git atual. Cada entrada remota é representada por um nome e uma URL. O nome da entrada remota é usado para referenciar outros comandos git, como o git fetch e git push.

Por exemplo, se o seu repositório Git tiver uma entrada remota chamada origin que aponta para o URL https://github.com/meu-usuário/meu-repositório.git, o comando git remote -v exibirá o seguinte:

origin https://github.com/meu-usuário/meu-repositório.git (fetch)

origin https://github.com/meu-usuário/meu-repositório.git (push)

Copiar código

**Desafio 6: Confira as mudanças nos arquivos.**

* No diretório do seu repositório local, digite no terminal o comando git status para ver arquivos modificados

**15O que aprendemos?**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139739/next)

**Nessa aula, você aprendeu como:**

* Baixar uma cópia de um repositório hospedado no GitHub para o seu computador, utilizando o comando git clone;
* Realizar alterações no código de um projeto e registrá-las com commits, utilizando os comandos git add e git commit;
* Visualizar quais arquivos foram modificados no repositório local, utilizando o comando git status;
* Listar os commits realizados no repositório, com dados do autor, data e mensagem de cada commit, utilizando o comando git log;
* Visualizar os repositórios remotos linkados com o repositório local, utilizando o comando git remote;
* Enviar commits feitos no repositório local para o repositório remoto, utilizando o comando git push;
* Baixar commits do repositório remoto para o repositório local, utilizando o comando git pull;
* Adicionar uma pessoa como colaboradora em um repositório no GitHub, e também como aceitar um convite de colaboração recebido.

**Transcrição**

**Gabrielle:** Como mencionamos anteriormente, estamos usando o Git apenas pelo terminal, digitando comandos. No entanto, existe uma forma mais fácil de usar esses comandos do Git por meio da interface gráfica, não é, Rodrigo?

**Rodrigo:** Isso mesmo! Existe a opção de usar o Git pelo prompt de comandos, seja o prompt do Windows ou o integrado no Visual Studio Code, mas como mencionamos antes, existe uma integração do próprio VS Code que facilita esse processo, de uma forma visual, com ícones. Portanto, vamos aprender a usar esse recurso.

Estou com o meu Visual Studio Code aberto e vou fechar essa janela do terminal clicando no ícone "X" no canto superior direito do terminal. Depois, na barra de menu, na lateral esquerda da janela, temos o ícone do Explorador, que é o primeiro, seguido do de Busca, que é o segundo.

O terceiro ícone, que tem forma de um Y com círculos nas extremidades, é, justamente, o ícone da integração do Git dentro do VS Code, chamada "*Source Control*" (Controle de Origem). Clicando nessa opção, abrimos uma aba onde mostra, de maneira visual, todas as alterações no código, e nós podemos fazer o commit, o pull, o push, tudo diretamente pela interface. Faremos uma simulação modificando o projeto.

Então, estou com o arquivo app.js aberto. Vamos imaginar que nós queremos fazer uma mudança no sistema. Imagine que as pessoas que estão usando nosso jogo achem que ficou difícil acertar um número entre 1 e 100, então alteramos para ser entre 1 e 50, para ficar um pouco mais fácil.

**let** listaDeNumerosSorteados = [];

**let** numeroLimite = 50;

**let** numeroSecreto = **gerarNumeroAleatorio**();

**let** tentativas = 1;

//código omitido

Copiar código

Após alterarmos o app.js, voltaremos ao explorador de arquivos e abriremos o index.html, onde trocaremos o texto do parágrafo na linha 23, dentro da <div class="container\_\_texto">, para Escolha um número entre 1 e 50. Pronto, fizemos uma mudança no projeto, alterando dois arquivos.

<!-- código omitido -->

<div class="container\_\_texto">

<h1>Adivinhe o <span class="container\_\_texto-azul">numero secreto</span></h1>

<p class="texto\_\_paragrafo">Escolha um número entre 1 e 50</p>

</div>

<!-- código omitido -->

Copiar código

Agora, no ícone do Git que fica na barra lateral esquerda da janela, notamos que ele está com o número 2 dentro de um círculo azul. Isso indica que há alterações no projeto.

**Gabrielle:** Esse número que aparece mostra que há mudanças e a quantidade delas.

**Rodrigo:** Ao clicarmos, a aba que abre à direita da barra mostra uma lista suspensa chamada "*Changes*" (Mudanças), que podemos minimizar ou expandir. Dentro dessa lista estão todos os arquivos que foram alterados.

Para fazermos um commit, ele até destaca na parte superior dessa aba uma caixa de texto, para digitarmos a mensagem, e um botão azul abaixo dela, escrito "Commit", para realizarmos o commit. Lembrando que, antes de fazermos o commit, temos que adicionar os arquivos.

Ao passarmos o mouse sobre cada um dos arquivos, ele mostra três ícones no lado direito do nome, sendo eles, da esquerda para direita:

* *Open file* (Abrir arquivo): clicando nele, abrimos o arquivo alterado. Esse ícone tem o formato de uma folha de papel com o canto superior direito dobrado e uma seta de retorno no canto superior esquerdo da folha;
* *Discard Changes* (Descartar mudanças): ao ser clicado, esse botão desfaz as mudanças no arquivo. Podemos clicar nele quando alteramos arquivos por engano. Ele tem o formato de uma seta curva que aponta para esquerda;
* *Stage Changes* (Preparar mudanças): clicando nele, adicionamos o arquivo ao próximo commit. O ícone é um sinal de mais.

Clicaremos no botão "*Stage Changes*" nos dois arquivos. Agora a lista "*Changes*" está vazia e os dois arquivos passaram para uma nova lista suspensa na parte superior chamada "*Staged Changes*" (Mudanças preparadas). São as alterações que já foram adicionadas e já podemos realizar o commit. Vamos fazer um commit, Gabi?

**Gabrielle:** Vamos lá! Para isso, precisamos informar uma mensagem novamente. Temos esse campo na parte superior da aba onde podemos escrever a mensagem do commit.

**Rodrigo:** Exatamente como fizemos no terminal, mas agora de uma maneira mais simples. Então, digitaremos a mensagem "Deixando o jogo mais fácil". Depois de escrever, clicamos no botão azul chamado commit.

A princípio, ele realizou o commit e o botão azul mudou o texto para "*Sync Changes 1*" (Sincronizar Mudanças 1), ou seja, se tornou um botão de sincronização. Ele detectou que há um commit no meu repositório local, mas que ele ainda não foi enviado para o GitHub.

Então, é exibido esse botão para sincronizar as mudanças com o repositório remoto. Ao clicarmos no botão "*Sync Changes 1*", os dados devem ser enviados para o GitHub. Para isso, apareceu um alerta no centro da tela com uma mensagem informando que essa ação sincroniza com o "origin/main".

Isso significa que ele enviará para o seu repositório remoto o que está na *branch* main. Esse alerta aparece em uma janela onde na parte inferior tem três botões:

* Ok: para confirmar;
* *Ok, Don't Show Again*" (Ok, Não mostre de novo): para confirmar a ação e não mostrar esse *pop-up* todas as vezes que fizermos um commit;
* *Cancel* (Cancelar): para cancelar o commit caso tenhamos clicado por engano.

Clicaremos no segundo botão, para confirmarmos o commit e essa mensagem não aparecer novamente. Após clicarmos, ele irá sincronizar as mudanças, o que pode demorar um pouco. Após concluir, o botão da aba do Git fica desativado e volta ao texto "Commit".

**Gabrielle:** Vamos conferir o repositório para confirmarmos se de fato ocorreu a sincronização?

**Rodrigo:** Recomendo que façamos isso. Então, abriremos o navegador na página do repositório no GitHub. Vamos atualizar a página, pressionando "F5" e, na parte superior da lista de arquivo, encontramos meu último *commit*.

rodrigoalura87 Deixando o jogo mais fácil

No canto superior direito da lista de arquivos, encontramos a informação que há três *commits*. Clicando nesse link, abrimos a lista de commits. Clicando no "Deixando o jogo mais fácil", acessamos o detalhamento desse *commit*, onde vemos uma demonstração visual das mudanças feitas. Portanto, funcionou.

A ideia é que nós sempre utilizemos o Git no VS Code por essa opção, pois simplifica o processo de sincronização das mudanças, certo, Gabi?

**Gabrielle:** Realmente, fica muito mais fácil. Nós não precisamos digitar todos aqueles comandos.

Estamos trabalhando de maneira colaborativa neste projeto e pode acontecer uma situação em que eu altere uma linha do nosso código e você altere exatamente a mesma linha. Como o GitHub se comporta quando isso acontece?

**Rodrigo:** É uma boa pergunta. Agora, não estou trabalhando sozinho, há várias pessoas trabalhando no projeto. Eventualmente, duas ou mais pessoas precisarão alterar um arquivo em comum.

Mesmo que as pessoas estejam trabalhando em funcionalidades distintas do sistema, pode ser que essas funcionalidades compartilhem o mesmo arquivo. E, caso duas pessoas alterem o mesmo arquivo, na mesma linha, isso causará um conflito.

Na sequência, veremos com calma como o Git funciona quando acontece essa situação de conflito.

**Transcrição**

**Gabrielle:** Ao trabalharmos com o Git e GitHub, é comum termos projetos nos quais várias pessoas colaboram, podendo ocorrer conflitos. Por exemplo, Rodrigo, eu fiz uma alteração no nosso projeto, alterando novamente o limite para 40. Vamos simular um conflito?

**Rodrigo:** Vamos verificar como o Git nos avisa quando há um conflito e o que ele faz. A Gabi já fez essa alteração no projeto, no repositório local dela, alterando os dados numéricos para 40.

Ela já fez o commit e o push para o GitHub. Para baixar esses commits, precisamos acessar a aba de Controle de Origem, ou seja, a integração com o Git. Em seguida, clicamos no terceiro ícone da barra lateral esquerda. Nessa aba, clicaremos no botão "*Views and More Actions*" (Views e Mais Ações), que tem o ícone de reticências (…) e fica no canto superior direito da aba.

Com isso, abrimos um menu suspenso onde a terceira opção é "*Pull*" (Puxar). Ao clicarmos nele, baixamos os commits da Gabi. Porém, antes disso, vamos alterar o mesmo arquivo na mesma linha e fazer um commit.

Quando o Git for tentar baixar o commit da Gabi, haverá um conflito com o meu commit, porque ambos estão alterando o mesmo arquivo, na mesma linha. Vamos descobrir o que acontece nessa situação.

Alteraremos o arquivo index.html, mudando a informação do parágrafo de 50 para 30. Depois alteramos o app.js, alterando o numeroLimite para 30.

Arquivo index.html

<!-- código omitido -->

<div class="container\_\_texto">

<h1>Adivinhe o <span class="container\_\_texto-azul">numero secreto</span></h1>

<p class="texto\_\_paragrafo">Escolha um número entre 1 e 30</p>

</div>

<!-- código omitido -->

Copiar código

Arquivo app.js

**let** listaDeNumerosSorteados = [];

**let** numeroLimite = 30;

**let** numeroSecreto = **gerarNumeroAleatorio**();

**let** tentativas = 1;

//código omitido

Copiar código

Após estas alterações, abriremos a aba do Git no VS Code, onde ele já identificou duas alterações. Adicionaremos os dois arquivos, clicando no ícone "+", e escreveremos uma mensagem deixando o jogo mais fácil ainda e clicaremos no botão "Commit".

Em seguida, clicaremos no botão com o ícone de reticências no canto superior direito da aba e, no menu, clicaremos no "*Pull*". O Git deveria identificar um conflito, ou seja, que meu repositório local foi alterado em uma linha específica e, no commit remoto, que ele tentará trazer, tem uma alteração na mesma linha do mesmo arquivo.

Ao clicarmos no Pull, ele tenta sincronizar e mostra uma mensagem de erro. Inclusive, apareceu um pop-up, no canto inferior direito do Visual Studio Code, informando que houve um conflito ao tentar fazer o *merge* (fusão), desses commits.

Não há problema em ter outros commits no repositório, o Git tenta fazer a sincronização automática. Porém, se houver um conflito no mesmo arquivo e na mesma linha, ele não consegue distinguir qual versão manter. Nem eu nem a Gabi poderemos decidir qual mudança será mantida.

O Git não escolhe uma versão, ele nos deixa decidir qual versão deve ser mantida. Por exemplo, no arquivo app.js ele mostra que uma pessoa desenvolvedora alterou o numeroLimite para 30 e outra alterou para 40. É nossa responsabilidade resolver.

Gabi, em seguida entenderemos como resolver o conflito e comunicar ao Git que o conflito foi resolvido e que está tudo certo no código.

**Transcrição**

**Gabrielle:** Tentamos realizar o *Pull* das alterações que estavam no nosso repositório remoto, para trazer essas alterações para o computador de Rodrigo, mas ocorreu um conflito. E agora aprenderemos a solucionar esse conflito.

**Rodrigo:** Estou com o arquivo index.html aberto com as marcações de conflito no texto do parágrafo, detectados pelo Git após eu fazer o *Pull* dos commits. O Git já detectou que houve um conflito e até mostra um botão azul, no canto inferior direito da tela, escrito "*Resolve in Merge Editor*". Se clicarmos nesse botão, poderemos resolver o conflito em uma ferramenta de edição de conflitos.

Você pode usar essa opção, então deixaremos um Para Saber Mais explicando como isso funciona. Outra opção é editar manualmente os arquivos. Vamos entender manualmente os arquivos para aprendermos como o Git nos sinaliza esse conflito.

Ao abrirmos os dois arquivos que apresentaram conflitos, que aparecem na lista "*Marge Changes*" dentro da aba de Controle de Origem, à esquerda. Eles têm um símbolo de exclamação na cor vermelha no canto direito do nome do arquivo, para indicar "*Há um conflito neste arquivo*", e isso acontece nos dois.

Ao abrirmos o app.js, ele fez uma marcação na linha onde declaramos a numeroLimite. Tem uma marcação em verde e outra em azul, basicamente para indicar a mudança de cada repositório.

A que eu fiz está em verde, inclusive com uma indicação de que é a mudança atual, ou seja, o valor é 30. O outro commit que eu tentei baixar está destacado em azul. A outra pessoa mudou a mesma linha, só que o valor que ela colocou foi 40.

Ele coloca sinais de maior que (>) ou menor que (<) antes dos nomes dos commits apenas para indicar visualmente no código onde está o conflito. Então vamos editar o arquivo, Gabi. Excluiremos essas linhas e definiremos qual das duas mudanças é a que será consolidada de fato.

**Gabrielle:** Isso acontece muito no dia a dia de uma pessoa desenvolvedora. Às vezes ocorre conflito em várias partes do código e você vai precisar conversar com as pessoas da sua equipe para alinhar qual será a alteração que será mantida e prosseguir com essa decisão.

**Rodrigo:** Vamos fazer isso agora. Vamos imaginar que eu e Gabi olhamos o log, e descobri que fiz uma alteração que deu conflito com o commit da Gabi. Conversei com ela e nós concordamos que Gabi ganhou. Perdi no par ou ímpar, então a mudança de 40 será a vencedora.

Como que eu digo para o Git: "*ignore essa linha do 30 e considera a de 40*", ou seja, desconsidera a minha e considera a do commit da Gabi? Podemos editar manualmente, excluindo essas linhas que geram conflito, assim como os comentários de origem do commit

Código app.js **antes**:

**let** listaDeNumerosSorteados = [];

<<<<<<< HEAD (**Current** **Change**)

**let** numeroLimite = 30;

=======

**let** numeroLimite = 40;

>>>>>>> ea448839319bea1248fFace4857aa327483a1b96 (**Incoming** **Change**)

**let** numeroSecreto = **gerarNumeroAleatorio**();

**let** tentativas = 1;

//código omitido

Copiar código

Código app.js **depois**:

**let** listaDeNumerosSorteados = [];

**let** numeroLimite = 40;

**let** numeroSecreto = **gerarNumeroAleatorio**();

**let** tentativas = 1;

//código omitido

Copiar código

Apagamos aquelas linhas, deixamos qual é a versão final, salvamos o arquivo. A mesma coisa no index.html. Ele indica a linha que está causando um conflito, então eu vou apagar o meu, que era o 30, e as linhas marcação do Git. Ficou apenas a Escolha número entre 1 e 40.

Código index.html **antes**:

<!-- código omitido -->

<div class="container\_\_texto">

<h1>Adivinhe o <span class="container\_\_texto-azul">numero secreto</span></h1>

<<<<<<< HEAD (Current Change)

<p class="texto\_\_paragrafo">Escolha um número entre 1 e 30</p>

=======

<p class="texto\_\_paragrafo">Escolha um número entre 1 e 40</p>

>>>>>>> ea448839319bea1248fFace4857aa327483a1b96 (Incoming Change)

</div>

<!-- código omitido -->

Copiar código

Código index.html **depois**:

<!-- código omitido -->

<div class="container\_\_texto">

<h1>Adivinhe o <span class="container\_\_texto-azul">numero secreto</span></h1>

<p class="texto\_\_paragrafo">Escolha um número entre 1 e 40</p>

</div>

<!-- código omitido -->

Copiar código

Salvei os dois arquivos após corrigi-los e apagar aquelas marcações do Git. Já está pronto, Gabi? O Git já sabe que o conflito foi resolvido?

**Gabrielle:** Precisamos agora realizar um commit que vai subir para o GitHub essa correção desse conflito que tivemos.

**Rodrigo:** Não basta apenas editarmos o arquivo e fazermos o ajuste. Nós precisamos avisar para o Git que o conflito foi resolvido. E esse registro, como foi mencionado pela Gabriela, é feito com um novo commit no projeto.

Então, após alterarmos os arquivos, clicaremos no ícone do Git, na barra lateral esquerda do VS Code. Será necessário fazer um novo commit, então clicaremos no botão *Stage Changes*, que tem o ícone "+" para adicionarmos os dois arquivos ao commit.

Após adicionarmos os dois arquivos, uma mensagem pré-definida é apresentada. A mensagem é: "*Merge branch 'main' of [link do repositório do GitHub]*". Podemos manter essa mensagem ou apagar e digitar outro texto, se desejarmos. Deixarei essa mensagem padrão.

Em seguida, clicaremos no botão "Commit". Agora o Git entendeu que nós resolvemos o conflito e aquele commit é o registro da resolução do conflito.

Atualmente, meu repositório local está atualizado, porém o botão "Commit" virou o botão de sincronização, pois esse novo commit não foi enviado para o GitHub. Portanto, nosso próximo passo é clicarmos nesse botão para fazermos a sincronização.

**Gabrielle:** Agora nós enviaremos tanto o commit que o Rodrigo realizou, representando aquela alteração, quanto o commit de correção desse conflito.

**Rodrigo:** Enviamos esse commit para o GitHub. Em princípio, a operação foi realizada sem problemas, mas vamos checar no site. Então, acessaremos a página do repositório no GitHub. Reparamos que a mensagem do último commit é justamente falando da mesclagem que resolve o conflito.

O repositório possui agora seis commits. Ao clicarmos no link dos commits, no canto superior direito, acessamos o histórico e notamos que ele mostra o commit de alteração do limite para 40, feita pela Gabriela. Também mostra o meu commit, representado pelo número 30, e o commit que resolveu esse conflito dos dois commits anteriores.

Então, é assim que funciona quando há um conflito: o Git marca no código, nós resolvemos o conflito fazendo um novo commit, e então tudo fica em ordem no repositório, certo?

**Gabrielle:** Exatamente. E às vezes, durante essa resolução de conflitos, partes do meu código e também partes do seu código serão mantidas, isso dependerá muito da situação.

**Rodrigo:** Com isso, aprendemos a lidar com um conflito. Na próxima aula aprenderemos sobre outras situações e recursos importantes do Git. O foco será no controle de versão, no histórico dos commits, e não somente nessa parte de colaboração com o GitHub.

**Transcrição**

**Gabrielle:** Tentamos realizar o *Pull* das alterações que estavam no nosso repositório remoto, para trazer essas alterações para o computador de Rodrigo, mas ocorreu um conflito. E agora aprenderemos a solucionar esse conflito.

**Rodrigo:** Estou com o arquivo index.html aberto com as marcações de conflito no texto do parágrafo, detectados pelo Git após eu fazer o *Pull* dos commits. O Git já detectou que houve um conflito e até mostra um botão azul, no canto inferior direito da tela, escrito "*Resolve in Merge Editor*". Se clicarmos nesse botão, poderemos resolver o conflito em uma ferramenta de edição de conflitos.

Você pode usar essa opção, então deixaremos um Para Saber Mais explicando como isso funciona. Outra opção é editar manualmente os arquivos. Vamos entender manualmente os arquivos para aprendermos como o Git nos sinaliza esse conflito.

Ao abrirmos os dois arquivos que apresentaram conflitos, que aparecem na lista "*Marge Changes*" dentro da aba de Controle de Origem, à esquerda. Eles têm um símbolo de exclamação na cor vermelha no canto direito do nome do arquivo, para indicar "*Há um conflito neste arquivo*", e isso acontece nos dois.

Ao abrirmos o app.js, ele fez uma marcação na linha onde declaramos a numeroLimite. Tem uma marcação em verde e outra em azul, basicamente para indicar a mudança de cada repositório.

A que eu fiz está em verde, inclusive com uma indicação de que é a mudança atual, ou seja, o valor é 30. O outro commit que eu tentei baixar está destacado em azul. A outra pessoa mudou a mesma linha, só que o valor que ela colocou foi 40.

Ele coloca sinais de maior que (>) ou menor que (<) antes dos nomes dos commits apenas para indicar visualmente no código onde está o conflito. Então vamos editar o arquivo, Gabi. Excluiremos essas linhas e definiremos qual das duas mudanças é a que será consolidada de fato.

**Gabrielle:** Isso acontece muito no dia a dia de uma pessoa desenvolvedora. Às vezes ocorre conflito em várias partes do código e você vai precisar conversar com as pessoas da sua equipe para alinhar qual será a alteração que será mantida e prosseguir com essa decisão.

**Rodrigo:** Vamos fazer isso agora. Vamos imaginar que eu e Gabi olhamos o log, e descobri que fiz uma alteração que deu conflito com o commit da Gabi. Conversei com ela e nós concordamos que Gabi ganhou. Perdi no par ou ímpar, então a mudança de 40 será a vencedora.

Como que eu digo para o Git: "*ignore essa linha do 30 e considera a de 40*", ou seja, desconsidera a minha e considera a do commit da Gabi? Podemos editar manualmente, excluindo essas linhas que geram conflito, assim como os comentários de origem do commit

Código app.js **antes**:

**let** listaDeNumerosSorteados = [];

<<<<<<< HEAD (**Current** **Change**)

**let** numeroLimite = 30;

=======

**let** numeroLimite = 40;

>>>>>>> ea448839319bea1248fFace4857aa327483a1b96 (**Incoming** **Change**)

**let** numeroSecreto = **gerarNumeroAleatorio**();

**let** tentativas = 1;

//código omitido

Copiar código

Código app.js **depois**:

**let** listaDeNumerosSorteados = [];

**let** numeroLimite = 40;

**let** numeroSecreto = **gerarNumeroAleatorio**();

**let** tentativas = 1;

//código omitido

Copiar código

Apagamos aquelas linhas, deixamos qual é a versão final, salvamos o arquivo. A mesma coisa no index.html. Ele indica a linha que está causando um conflito, então eu vou apagar o meu, que era o 30, e as linhas marcação do Git. Ficou apenas a Escolha número entre 1 e 40.

Código index.html **antes**:

<!-- código omitido -->

<div class="container\_\_texto">

<h1>Adivinhe o <span class="container\_\_texto-azul">numero secreto</span></h1>

<<<<<<< HEAD (Current Change)

<p class="texto\_\_paragrafo">Escolha um número entre 1 e 30</p>

=======

<p class="texto\_\_paragrafo">Escolha um número entre 1 e 40</p>

>>>>>>> ea448839319bea1248fFace4857aa327483a1b96 (Incoming Change)

</div>

<!-- código omitido -->

Copiar código

Código index.html **depois**:

<!-- código omitido -->

<div class="container\_\_texto">

<h1>Adivinhe o <span class="container\_\_texto-azul">numero secreto</span></h1>

<p class="texto\_\_paragrafo">Escolha um número entre 1 e 40</p>

</div>

<!-- código omitido -->

Copiar código

Salvei os dois arquivos após corrigi-los e apagar aquelas marcações do Git. Já está pronto, Gabi? O Git já sabe que o conflito foi resolvido?

**Gabrielle:** Precisamos agora realizar um commit que vai subir para o GitHub essa correção desse conflito que tivemos.

**Rodrigo:** Não basta apenas editarmos o arquivo e fazermos o ajuste. Nós precisamos avisar para o Git que o conflito foi resolvido. E esse registro, como foi mencionado pela Gabriela, é feito com um novo commit no projeto.

Então, após alterarmos os arquivos, clicaremos no ícone do Git, na barra lateral esquerda do VS Code. Será necessário fazer um novo commit, então clicaremos no botão *Stage Changes*, que tem o ícone "+" para adicionarmos os dois arquivos ao commit.

Após adicionarmos os dois arquivos, uma mensagem pré-definida é apresentada. A mensagem é: "*Merge branch 'main' of [link do repositório do GitHub]*". Podemos manter essa mensagem ou apagar e digitar outro texto, se desejarmos. Deixarei essa mensagem padrão.

Em seguida, clicaremos no botão "Commit". Agora o Git entendeu que nós resolvemos o conflito e aquele commit é o registro da resolução do conflito.

Atualmente, meu repositório local está atualizado, porém o botão "Commit" virou o botão de sincronização, pois esse novo commit não foi enviado para o GitHub. Portanto, nosso próximo passo é clicarmos nesse botão para fazermos a sincronização.

**Gabrielle:** Agora nós enviaremos tanto o commit que o Rodrigo realizou, representando aquela alteração, quanto o commit de correção desse conflito.

**Rodrigo:** Enviamos esse commit para o GitHub. Em princípio, a operação foi realizada sem problemas, mas vamos checar no site. Então, acessaremos a página do repositório no GitHub. Reparamos que a mensagem do último commit é justamente falando da mesclagem que resolve o conflito.

O repositório possui agora seis commits. Ao clicarmos no link dos commits, no canto superior direito, acessamos o histórico e notamos que ele mostra o commit de alteração do limite para 40, feita pela Gabriela. Também mostra o meu commit, representado pelo número 30, e o commit que resolveu esse conflito dos dois commits anteriores.

Então, é assim que funciona quando há um conflito: o Git marca no código, nós resolvemos o conflito fazendo um novo commit, e então tudo fica em ordem no repositório, certo?

**Gabrielle:** Exatamente. E às vezes, durante essa resolução de conflitos, partes do meu código e também partes do seu código serão mantidas, isso dependerá muito da situação.

**Rodrigo:** Com isso, aprendemos a lidar com um conflito. Na próxima aula aprenderemos sobre outras situações e recursos importantes do Git. O foco será no controle de versão, no histórico dos commits, e não somente nessa parte de colaboração com o GitHub.

**04Resolvendo conflitos com Visual Studio Code Merge Editor**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/144180/next)

Quando você está trabalhando em um projeto grande, principalmente se muitas pessoas estão envolvidas, é comum aparecerem os chamados "conflitos". Mas o que são esses conflitos?

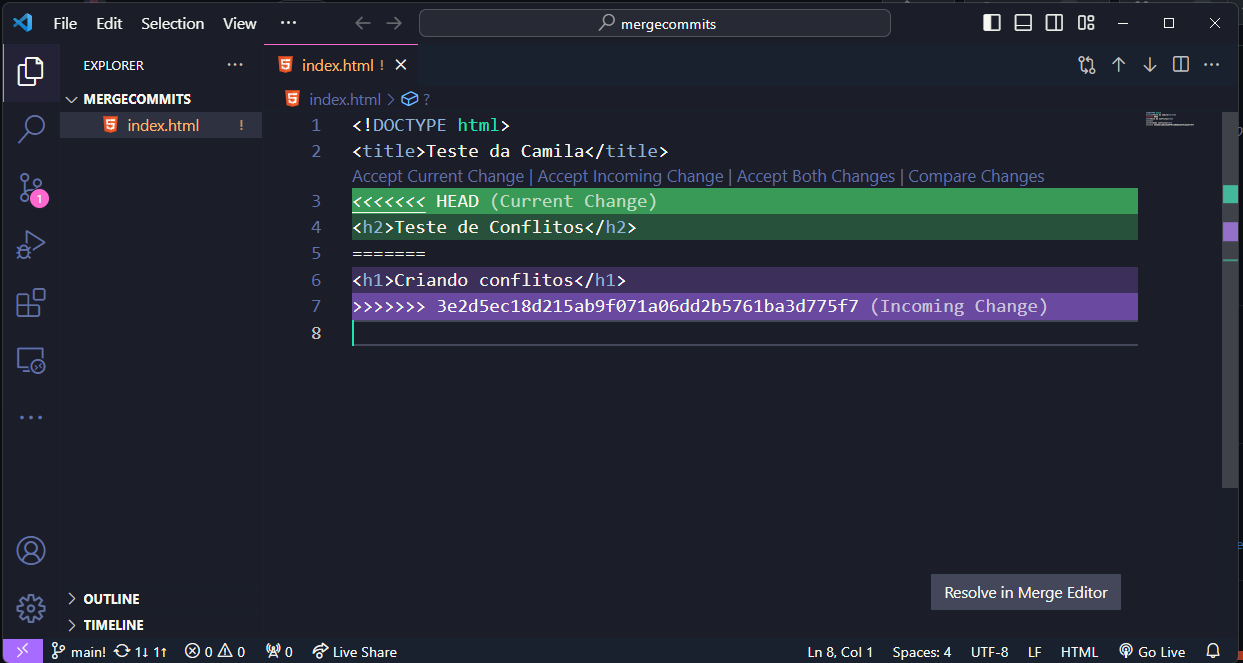
Imagine que o projeto é um grande quebra-cabeça e cada pessoa está trabalhando em uma parte dele. Se duas pessoas tentarem encaixar peças diferentes no mesmo lugar, surge um conflito. No mundo do desenvolvimento de software, isso ocorre quando duas pessoas editam o mesmo pedaço de código de formas diferentes.

Há várias formas de resolver conflitos, podemos utilizar a linha de comando ou o próprio Visual Studio Code. O editor de código fornece mais de uma forma para resolver conflitos de mesclagem entre o código local (o que está na sua máquina) e o remoto(o que está no github, por exemplo).

Uma possibilidade é utilizar a ferramenta **“Merge Editor”.** Vamos conferir seu funcionamento?

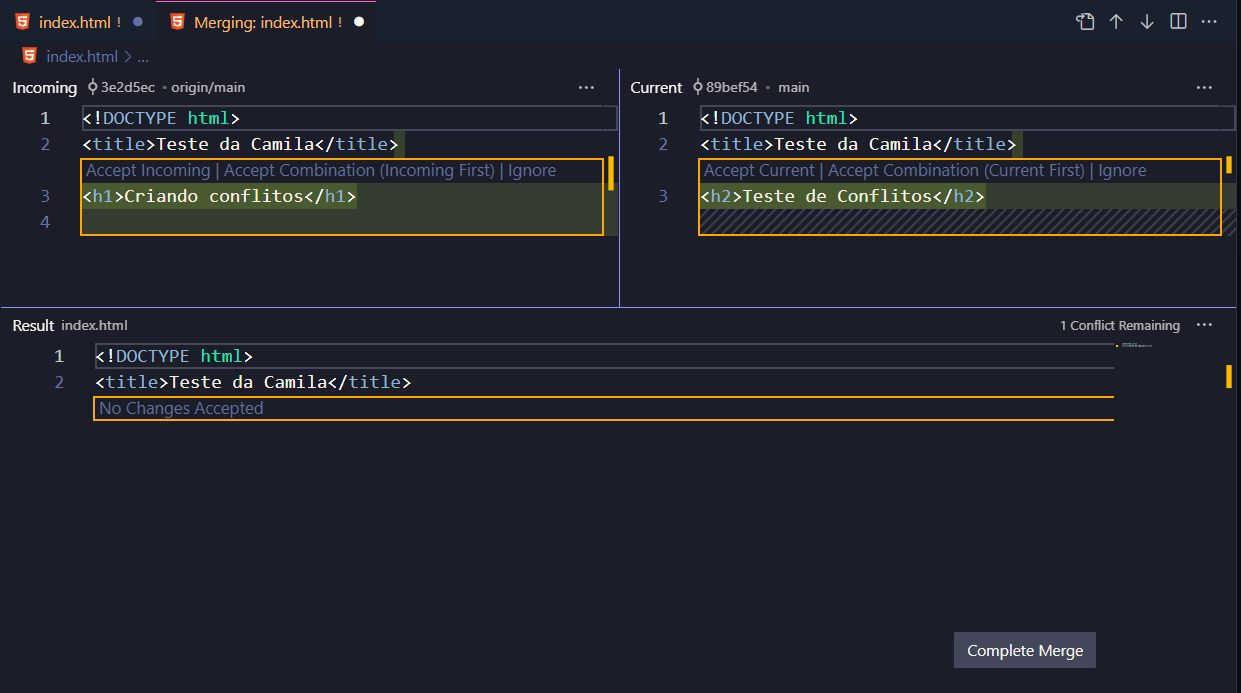
**Resolvendo conflitos no Merge Editor**

No exemplo a seguir, nós temos duas versões de um código na branch main: uma no repositório do github e outra modificação diferente no ambiente local. Ao realizarmos o git -push para a branch main, ocorreu um **conflito** e precisamos resolvê-lo para que a atualização suba para o repositório no github corretamente, como a imagem nos apresenta:

Para solucionarmos o problema, clicamos na opção “Resolve in Merge Editor”, como no print abaixo:



Após o clique, somos redirecionados para outra aba que apresenta as modificações no arquivo, vamos entender o que cada opção significa:



A tela do Visual Studio Code está dividida em três partes:

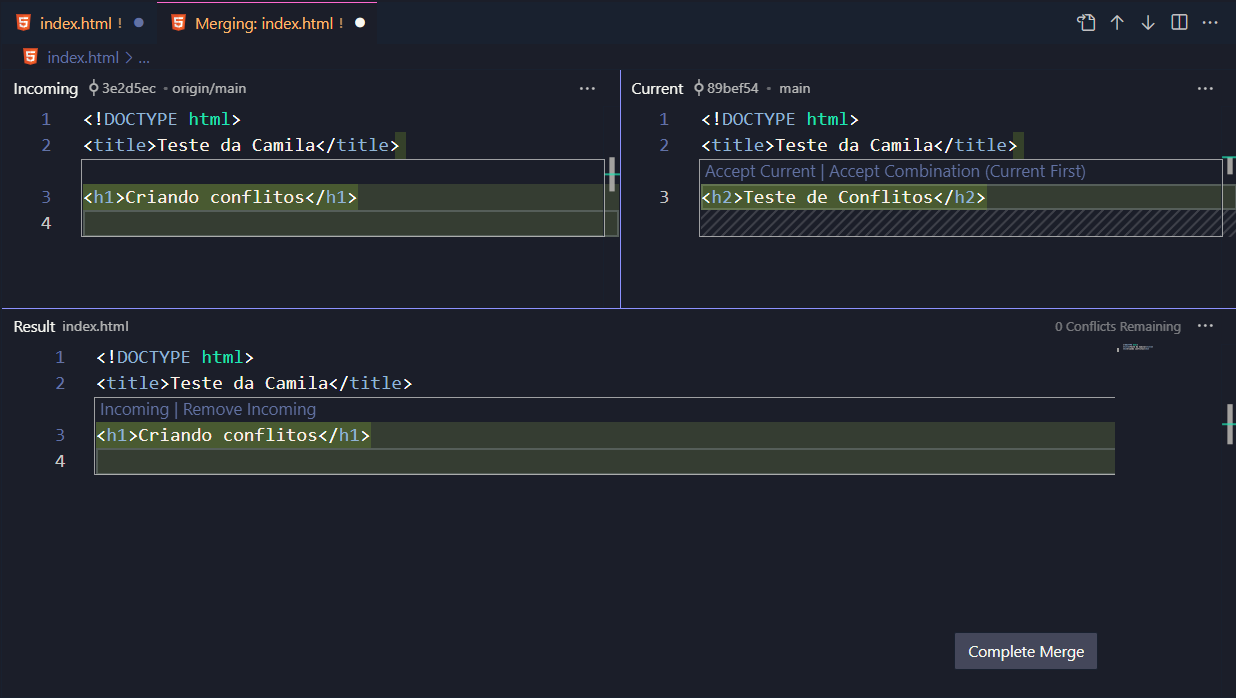
* **Incoming (remoto)**: modificações que chegam do repositório remoto.
* **Current (local)**: modificações locais.
* **Result (resultado)**: resultado do merge, ou seja, a resolução dos conflitos de mesclagem. É o estado atual do arquivo.

Os quadrados na cor amarela em volta do código no campo “Incoming” e “Current” são **marcadores de conflito**: exibem o conteúdo que apresenta conflito no arquivo.

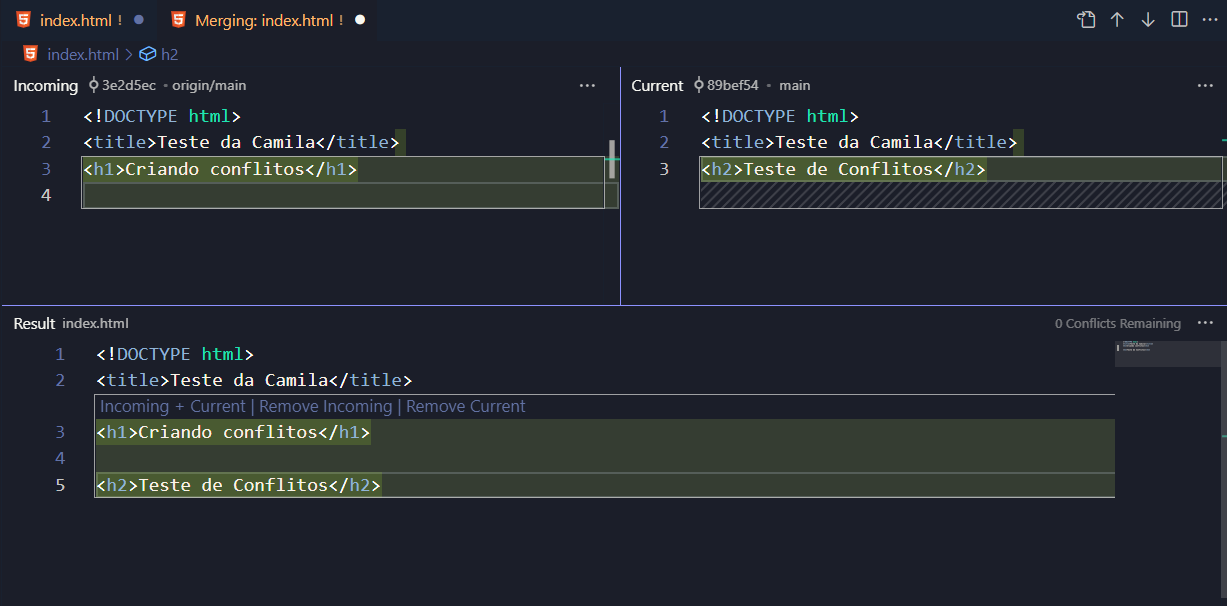
**Campo Incoming**

**->** No **campo “Incoming”,** acima da linha de código dos marcadores de conflito no campo há outras opções que resultam na alteração do código atual:

* **Accept Incoming**: aceita modificações oriundas do **remoto**



* **Accept Combination Incoming First**: realiza a combinação com as linhas do código do repositório remoto no topo.

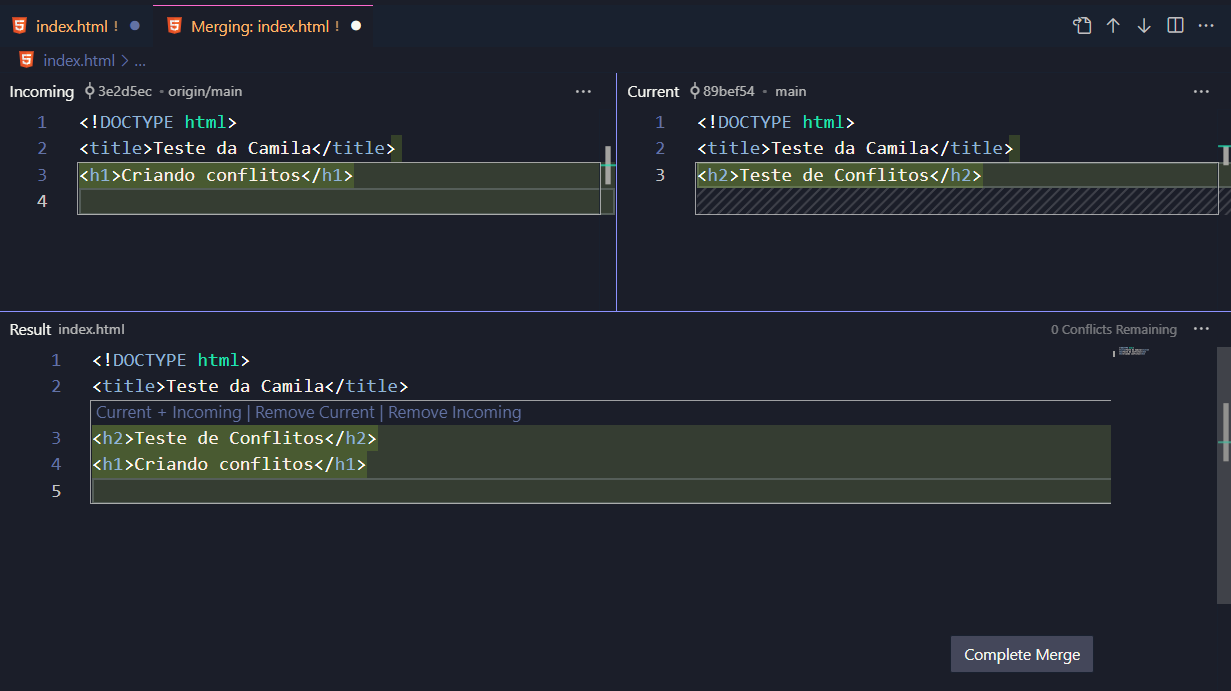


* **Ignore:** ignora as modificações.

**Campo Current**

**->** O campo **“Current”** trabalha com as modificações locais do documento.

* **Accept Current**: Aceita a modificação local no resultado do documento
* **Accept combination Current First**: Aceita a combinação local + remoto. Nos resultados a linha de código com a tag <h2> fica antes de <h1>, comprovando que o código local é inserido primeiro que o remoto.



* **Ignore**: ignora as modificações no resultado no final.

**Após selecionarmos a opção com o resultado desejado, devemos:**

1. Salvar o arquivo
2. Clicar no botão “complete Merge”
3. Realizar o commit das modificações
4. Sincronizar as modificações realizando o push.

**Para saber mais:**

* [Como o Visual Studio facilita o controle de versão com o Git](https://learn.microsoft.com/pt-br/visualstudio/version-control/git-with-visual-studio?view=vs-2022)
* [Como resolver conflitos de mesclagem no Visual Studio](https://learn.microsoft.com/pt-br/visualstudio/version-control/git-resolve-conflicts?view=vs-2022)
* [Um guia muito útil para mesclar conflitos - em Inglês](https://www.youtube.com/watch?v=HosPml1qkrg)

**06Faça como eu fiz: utilizando o Git na IDE**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139773/next)

Agora é com você! Faça alguma alteração no projeto e a envie para o repositório remoto no GitHub usando a integração do Git no VS Code ao invés do terminal, conforme realizado na aula.

Ver opinião do instrutor

**Opinião do instrutor**

Com o seu projeto aberto no VS Code, altere alguma parte do código e salve o projeto.

Clique no botão lateral esquerdo de *Source Control* para abrir a integração Git no VS Code. Lá será mostrado quais arquivos contém alterações.

Adicione cada um dos arquivos, clicando no botão de *+* ao lado do nome de cada um.

Escreva uma mensagem de commit, no devido campo correspondente, descrevendo a alteração feita. Por fim, clique no botão *Sync Changes* para enviar as alterações para o repositório remoto no GitHub.

**07Desafios: hora da prática**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/145910/next)

Conflitos em diferentes versões no código é algo comum no cotidiano da pessoa desenvolvedora e é essencial saber como resolver quando esses conflitos aparecem. A ideia do próximo desafio é gerar e resolver um conflito entre um repositório remoto e local, vamos praticar com os desafios não obrigatórios?

**Desafios**

* Crie um novo repositório local
* Adicione o repositório remoto criado nos exercícios anteriores ao seu repositório local.
* Faça uma alteração no repositório local e envie para o remoto.
* Resolva os conflitos manualmente, escolhendo quais alterações serão mantidas
* Realize um commit para registrar a resolução do conflito.
* Verifique quais arquivos foram adicionados
* Sincronize o repositório local com o repositório remoto no GitHub.

Caso precise de ajuda, opções de solução das atividades estarão disponíveis na seção “Opinião da pessoa instrutora”.

**08O que aprendemos?**

[**Próxima Atividade**](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139774/next)

**Nessa aula, você aprendeu como:**

* Utilizar o Git pela integração do VSCode, ao invés de utilizar pelo terminal;
* Entender como acontecem conflitos de códigos em commits que modificam um mesmo arquivo, feitos por pessoas distintas;
* O Git sinaliza no código um conflito, via marcações visuais;
* Resolver um conflito manualmente, editando o arquivo e realizando um commit que marca o conflito como resolvido.
* **Transcrição**
* **Rodrigo:** Bom, Gabi, já aprendemos vários recursos do GitHub e do Git. Contudo, até agora, focamos mais na parte de **colaboração**.
* Criei um repositório no meu usuário do Git, você fez o clone, te adicionei como colaboradora e começamos a trabalhar simulando **rotinas** de desenvolvimento de software. Você realizou mudanças no código, *commits*, assim como eu no meu computador, baixando os seus commits e enviando os commits para o GitHub.
* Cada commit representa uma **versão** do código, que fica registrada no histórico. Conhecemos o git-log e conseguimos conferir todo o histórico de versões. E, eventualmente, vamos querer voltar no tempo, pois alguém pode pedir para **desfazer uma alteração**. O Git também nos ajuda com esta situação, correto?
* **Gabrielle:** Sim, o Git pode nos ajudar nessas situações. Notei que você fez uma nova modificação no nosso projeto. Havia a imagem de uma menina ao fundo e você a removeu. Agora, vamos supor que queremos trazer essa imagem de volta, retornar para a versão em que tínhamos essa imagem no projeto. Como podemos fazer isso?
* **Rodrigo:** Neste caso, poderíamos entrar no site do GitHub, clicar na lista do log para ver os commits e encontrar qual foi o commit que fez essa alteração. Lembre-se de que é possível detalhar o que aquele commit fez no código e quais arquivos foram modificados.
* Neste nosso exemplo, seria simples encontrar as alterações, porque a mudança ocorreu apenas no index.html. Porém, e se o commit tivesse alterado 86 arquivos, por exemplo? Não seria nada produtivo ter que verificar arquivo por arquivo, ir ao VS Code e desfazer a alteração.
* Presumo que o Git tenha um comando para automatizar esse processo, certo, Gabi?
* **Gabrielle:** Isso mesmo. Vamos descobrir qual é esse comando?
* Vamos abrir o VS Code e, naquela ferramenta do GitHub que estávamos utilizando no próprio VS Code, para abrir o log, aquele histórico dos nossos commits. A ferramenta é a "Source Control", no terceiro botão do menu lateral esquerdo.
* Clicando nessa ferramenta, temos o menu de três pontos (botão "Views and More Actions"). Vamos clicar nele e verificar se temos a opção de log. Procurando nas opções, não vemos nenhuma que mostre nosso histórico de commits.
* **Rodrigo:** Verdade. Começamos a usar essa integração do Visual Studio Code com o Git, mas nem todos os comandos estão disponíveis via menu. Os mais comuns, como ver o status, qual arquivo foi modificado, fazer um commit, *ADD*, *push*, *pull*, são exibidos.
* Mas, se quisermos executar um log ou outros comandos mais avançados, e que não são tão comuns, o VS Code não mostra. Sendo assim, teremos que usar o terminal novamente.
* **Gabrielle:** Vamos abrir novamente o terminal, clicando em "Terminal > New Terminal" no menu superior. Vamos minimizar o menu lateral esquerdo para deixar apenas o terminal na tela.
* Agora, vamos rodar o comando git log, que já conhecemos. Como vimos anteriormente, esse comando traz algumas informações dos nossos commits. Cada commit possui um **ID**, o qual identifica esse commit de maneira única.
* **Rodrigo:** O ID único é a maneira pela qual o Git diferencia cada commit. É como se fosse um ID de registro no banco de dados. Precisamos saber qual é o commit que queremos desfazer e, para isso, usaremos esse ID.
* **Gabrielle:** O commit que queremos desfazer é o "**removendo foto**", então vamos copiar o ID dele (o código de letras e números ao lado de "commit").
* **commit** 2ad48c068dc9677fb57efec70620700410f976b0
* Copiar código
* Em seguida, pressionamos a tecla "Q" para retornar ao nosso terminal. **O comando para reverter um commit é o git revert**, passando ao lado o ID do commit a ser desfeito.
* git revert 2ad48c068dc9677fb57efec70620700410f976b0
* Copiar código
* **Rodrigo:** Executando o comando git revert, é aberta uma pequena janela no topo da tela. Isso acontece porque o git revert é o comando responsável por realizar o processo de identificar o commit pelo ID passado como parâmetro, analisar cada alteração feita por ele e desfazer essas alterações sozinho.
* Entretanto, o revert não desfaz as alterações modificando apenas os arquivos. Ele cria um novo commit para registrar esse revert.
* **Gabrielle:** Portanto, o revert já nos fornece uma **mensagem** para esse novo commit,dizendo que realizou um revert e traz a mensagem do commit original, por exemplo, além do ID desse commit.
* *Revert "removendo foto"*
* *This reverts commit 2ad48c068dc9677fb57efec70620700410f976b0.*
* Fechando esta janela, essa mensagem já ficará salva para o novo commit.
* Agora, vamos executar o comando git log no terminal para verificar o que aconteceu.
* Na lista de commits, foi retornado esse novo commit no topo da lista, indicando que ocorreu um revert do commit de "removendo foto".
* **Rodrigo:** Então, na verdade, o revert não apaga o commit original. O que ele faz é criar um novo commit que efetua uma espécie de "Ctrl + Z" no commit original. Todos os arquivos modificados pelo commit original têm suas alterações desfeitas, mas o revert fica registrado no histórico.
* No entanto, como é um novo commit, ele está apenas no nosso repositório local. Portanto, quisermos enviá-lo para o GitHub, será necessário sincronizar, rodando novamente o seguinte comando no terminal:
* git push origin main
* Copiar código
* Com isso,sincronizamos e enviamos esse commit de revert para o repositório remoto, permitindo que outras pessoas possam baixá-lo.
* Será que essa alteração funcionou? Vamos verificar no site se a foto da pessoa foi restaurada
* **Gabrielle:** Abrindo o navegador, podemos notar que a imagem da menina já está no plano de fundo novamente. Portanto, deu certo!
* **Rodrigo:** Nós fizemos o revert, desfizemos a alteração de um determinado commit. O próprio git se encarregou de olhar cada arquivo, desfazer as respectivas alterações e registrar isso no histórico, revertendo o código para a versão anterior.
* Esta é uma situação comum. Eventualmente, as pessoas que usam o sistema podem solicitar uma alteração e, depois de uma semana ou um mês, elas podem mudar de ideia e dizer:
* — Olha, testamos a alteração que você fez naquele dia, e não era exatamente o que queríamos. Você pode reverter para o como estava antes?
* E nós não lembraremos de cabeça como era o código anterior. Portanto, graças a essa funcionalidade de registro por commits, nós conseguimos localizar este commit e fazer o revert, recuperando a versão do código daquele momento.
* **Gabrielle:** O histórico de comandos pode ser útil em **outras situações**. Por exemplo, vamos imaginar que em vez de querer reverter um commit, tenhamos feito algo no repositório local e queiramos **excluir** esse commit, removê-lo do histórico. É possível fazer isso?
* **Rodrigo:** Essa é uma situação diferente. Agora não queremos mais apenas desfazer um commit, mas de fato apagá-lo. É possível fazer isso, e aprenderemos a seguir.